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

LNCS 7204

Parallel Processing and Applied Mathematics

9th International Conference, PPAM 2011
Torun, Poland, September 2011
Revised Selected Papers, Part II

2 Part II

Springer

Lecture Notes in Computer Science 7204

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

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

Parallel Processing and Applied Mathematics

9th International Conference, PPAM 2011
Torun, Poland, September 11-14, 2011
Revised Selected Papers, Part II

Springer

Volume Editors

Roman Wyrzykowski
Czestochowa University of Technology, Poland
E-mail: roman@icis.pcz.pl

Jack Dongarra
University of Tennessee, Knoxville, TN, USA
E-mail: dongarra@cs.utk.edu

Konrad Karczewski
Czestochowa University of Technology, Poland
E-mail: xeno@icis.pcz.pl

Jerzy Waśniewski
Technical University, Kongens Lyngby, Denmark
E-mail: jw@imm.dtu.dk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-31499-5 e-ISBN 978-3-642-31500-8
DOI 10.1007/978-3-642-31500-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012941360

CR Subject Classification (1998): D.2, H.4, D.4, C.2.4, D.1.3, H.3, F.2

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume comprises the proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics – PPAM 2011, which was held in Toruń, Poland, September 11–14, 2011. It was organized by the Department of Computer and Information Science of the Częstochowa University of Technology, with the help of the Nicolaus Copernicus University in Toruń, Faculty of Mathematics and Computer Science. The main organizer was Roman Wyrzykowski.

PPAM is a biennial conference. Eight previous events have been held in different places in Poland since 1994. The proceedings of the last five 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; Wrocław, 2009, vols. 6067 and 6068).

The PPAM conferences have become an international forum for exchanging ideas between researchers involved in scientific and parallel computing, including theory and applications, as well as applied and computational mathematics. The focus of PPAM 2011 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, and cloud computing.

This meeting gathered more than 200 participants from 33 countries. A strict refereeing process resulted in acceptance of 130 contributed presentations, while approximately 45% 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 non-numerical algorithms
- Tools and environments for parallel/distributed/grid computing
- Applications of parallel/distributed computing
- Applied mathematics, neural networks and evolutionary computing
- History of computing

The plenary and invited talks were presented by:

- David A. Bader from the Georgia Institute of Technology (USA)
- Paolo Bientinesi from the RWTH Aachen (Germany)
- Christopher Carothers from the Rensselaer Polytechnic Institute (USA)
- Ewa Deelman from the University of Southern California (USA)
- Jack Dongarra from the University of Tennessee and Oak Ridge National Laboratory (USA)
- Geoffrey Ch. Fox from the Indiana University (USA)
- Fred Gustavson from the Umeå University (Sweden) and emeritus from the IBM T.J. Watson Research Center (USA)

- Tony Hey from the Microsoft Research
- Bo Kågström from the Umeå University (Sweden)
- Jakub Kurzak from the University of Tennessee (USA)
- Jarek Nabrzyski from the University of Notre Dame (USA)
- Raymond Namyst from the University of Bordeaux & INRIA (France)
- Victor Pankratius from the University of Karlsruhe (Germany)
- Markus Pueschel from the ETH Zurich (Switzerland)
- Eugen Schenfeld from the IBM T.J. Watson Research Center (USA)
- Robert Strzodka from the Max Planck Institut für Informatik (Germany)
- Bolesław Szymański from the Rensselaer Polytechnic Institute (USA)
- Richard W. Vuduc from the Georgia Institute of Technology (USA)
- Jerzy Waśniewski from the Technical University of Denmark (Denmark)

Important and integral parts of the PPAM 2011 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)
- Minisymposium on Autotuning organized by Richard W. Vuduc from the Georgia Institute of Technology (USA) and Roman Wyrzykowski from the Częstochowa University of Technology (Poland)
- Workshop on Memory and Data Parallelism on Multi- and Manycore Platforms organized by Michael Bader from the University of Stuttgart (Germany), Carsten Trinitis, and Josef Weidendorfer from the TU München (Germany)
- Workshop on Models, Algorithms and Methodologies for Hierarchical Parallelism in New HPC Systems organized by Giulliano Laccetti and Marco Lapegna from the University of Naples Federico II (Italy) and Raffaele Montella from the University of Naples "Parthenope" (Italy)
- Workshop on Scheduling for Parallel Computing—SPC 2011—organized by Maciej Drozdowski from the Poznań University of Technology (Poland)
- The 4th Workshop on Language-Based Parallel Programming Models—WLPP 2011—organized by Ami Marowka from the Bar-Ilan University (Israel)
- The Second Workshop on Scalable Computing in Distributed Systems and the 7th Workshop on Large-Scale Computations on Grids—ScoDiS-LaSCoG 2011—organized by Dana Petcu from the West University of Timisoara (Romania) and Marcin Paprzycki from WSM and the Systems Research Institute of the Polish Academy of Sciences (Poland)
- The Third Workshop on Performance Evaluation of Parallel Applications on Large-Scale Systems organized by Jan Kwiatkowski from the Wrocław University of Technology (Poland)
- Workshop on Parallel Computational Biology—PBC 2011—organized by David A. Bader from the Georgia Institute of Technology (USA), Jarosław Żola from the Iowa State University (USA), and Scott Emrich from the University of Notre Dame (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 Vilnius University (Lithuania)
- Minisymposium on High-Performance Computing Interval Methods organized by Bartłomiej J. Kubica from the Warsaw University of Technology (Poland)
- Workshop on Complex Colective Systems organized by Paweł Topa and Jarosław Wąs from the AGH University of Science and Technology in Cracow (Poland)
- The First Workshop on Service-Oriented Architecture in Distributed Systems—SOADS 2011—organized by Jan Kwiatkowski from the Wrocław University of Technology (Poland) and Dariusz Wawrzyniak from the Poznań University of Technology (Poland)

The PPAM 2011 meeting began with five tutorials:

- Scientific Computing with GPUs, by Dominik Göddeke from the University of Dortmund (Germany), Jakub Kurzak from the University of Tennessee (USA), Jan-Philipp Weiss from the Karlsruhe Institute of Technology (Germany), as well as André Heidekrüger from AMD, and Tim Schröder from NVIDIA
- StarPU System for Heterogeneous Multicore Architectures, by Raymond Namyst from the University of Bordeaux and INRIA (France)
- Tutorial on the 100th Anniversary of Cholesky's Algorithm, by Fred Gustavson from the Umeå University (Sweden) and emeritus from the IBM T.J. Watson Research Center (USA) and Jerzy Waśniewski from the Technical University of Denmark (Denmark)
- FutureGrid, by Geoffrey Ch. Fox from the Indiana University (USA)
- Best Practices to Run Applications in HPC Environments, by the POWIEW Project team (Poland)

The PPAM Best Poster Award is granted to the best poster on display at the PPAM conferences, and was established 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 2011 winners were Damian Wóicik, Marcin Kurowski, Bogdan Rosa, and Michał Ziemiański from the Institute of Meteorology and Water Management in Warsaw, who presented the poster "A Study on Parallel Performance of the EULAG F90/95 Code."

The Special Award was bestowed to Andrzej Jarynowski from the Jagiellonian University and Przemysław Gawroński, Krzysztof Kułakowski from the AGH University of Science and Technology in Kraków, who presented the poster "How the Competitive Altruism Leads to Bistable Homogeneous States of Cooperation or Defection."

Automated Performance Tuning ("Autotuning") of Software: The complexity of modern machines makes performance tuning a tedious and time-consuming task. The goal of *autotuning* techniques is to automate the process of selecting the highest-performing program implementation from among a space of candidates, guided by experiments. An experiment is the execution of a benchmark and observation of its performance; such experiments may be used directly to test a candidate implementation, or may be used to calibrate a model that is then used to select such an implementation. Roughly speaking, autotuning research considers questions of how to identify and generate the space of candidate program implementations as well as how to find (or search for) the best implementation given such a space. A system that implements an autotuning process is an *autotuner*. An autotuner may be a stand-alone code generation system or may be part of a compiler.

The Minisymposium on Autotuning featured a number of invited and contributed talks covering recent and diverse advances, including:

- A new high-level rewrite system for linear algebra computations, with applications to computational physics and biology (by P. Bientinesi)
- Novel uses of machine learning to facilitate searching (M. Püschel)
- The extension of autotuning ideas into general software engineering processes, such as tuning the software architecture (V. Pankratius)
- New code generation and search space pruning techniques for dense linear algebra targeted at GPU architectures (J. Kurzak and H.H.B. Sørensen)
- Reducing tuning time for high-performance LINPACK using novel performance models (P. Łuszczek)

The organizers are indebted to the PPAM 2011 sponsors, whose support was vital to the success of the conference. The main sponsor was the Intel Corporation. The other sponsors were: IBM Corporation, Hewlett-Packard Company, Microsoft Corporation, and AMD. We thank 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 the Nicolaus Copernicus University in Toruń, 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 Andrzej Rozkosz, and Piotr Bała from the Nicolaus Copernicus University.

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 2013, which will be held during September 8–11, 2013, in Warsaw, the capital of Poland.

February 2012

Roman Wyrzykowski
Jack Dongarra
Konrad Karczewski
Jerzy Waśniewski

Organization

Program Committee

Węglarz, Jan	Poznań University of Technology, Poland Honorary Chair
Wyrzykowski, Roman	Częstochowa University of Technology, Poland Program Committee Chair
Szymański, Bolesław	Rensselaer Polytechnic Institute, USA Program Committee Vice-chair
Arbenz, Peter	ETH, Zurich, Switzerland
Bała, Piotr	Nicolaus Copernicus University, Poland
Bader, David A.	Georgia Institute of Technology, USA
Bader, Michael	University of Stuttgart, Germany
Blaheta, Radim	Institute of Geonics, Czech Academy of Sciences
Błażewicz, Jacek	Poznań University of Technology, Poland
Bokota, Adam	Częstochowa University of Technology, Poland
Bouvry, Pascal	University of Luxembourg
Burczyński, Tadeusz	Silesia University of Technology, Poland
Brzeziński, Jerzy	Poznań University of Technology, Poland
Bubak, Marian	Institute of Computer Science, AGH, Poland
Čiegis, Raimondas	Vilnius Gediminas Technical University, Lithuania
Clematis, Andrea	IMATI-CNR, Italy
Cunha, Jose	University New of Lisbon, Portugal
Czech, Zbigniew	Silesia University of Technology, Poland
Deelman, Ewa	University of Southern California, USA
Dongarra, Jack	University of Tennessee and ORNL, USA
Drozdowski, Maciej	Poznań University of Technology, Poland
Elmroth, Erik	Umea University, Sweden
Flasiński, Mariusz	Jagiellonian University, Poland
Ganzha, Maria	IBS PAN, Warsaw, Poland
Gepner, Pawel	Intel Corporation
Gondzio, Jacek	University of Edinburgh, Scotland, UK
Gościński, Andrzej	Deakin University, Australia
Grigori, Laura	INRIA, France
Grzech, Adam	Wroclaw University of Technology, Poland
Guinand, Frederic	Université du Havre, France
Herrero, José R.	Universitat Politècnica de Catalunya, Barcelona, Spain
Hluchy, Ladislav	Slovak Academy of Sciences, Bratislava

Jakl, Ondrej	Institute of Geonics, Czech Academy of Sciences
Janciak, Ivan	University of Vienna, Austria
Jeannot, Emmanuel	INRIA, France
Kalinov, Alexey	Cadence Design System, Russia
Kamieniarz, Grzegorz	A. Mickiewicz University, Poznań, Poland
Kiper, Ayse	Middle East Technical University, Turkey
Kitowski, Jacek	Institute of Computer Science, AGH, Poland
Korbicz, Józef	University of Zielona Góra, Poland
Kozielski, Stanislaw	Silesia University of Technology, Poland
Kranzlmueller, Dieter	Ludwig Maximilian University, Munich, and Leibniz Supercomputing Centre, Germany
Krawczyk, Henryk	Gdańsk University of Technology, Poland
Krzyżanowski, Piotr	University of Warsaw, Poland
Kwiatkowski, Jan	Wrocław University of Technology, Poland
Laccetti, Giulliano	University of Naples Federico II, Italy
Lapegna, Marco	University of Naples Federico II, Italy
Lastovetsky, Alexey	University College Dublin, Ireland
Maksimov, Vyacheslav I.	Ural Branch, Russian Academy of Sciences
Malyshkin, Victor E.	Siberian Branch, Russian Academy of Sciences
Margalef, Tomas	Universitat Autonoma de Barcelona, Spain
Margenov, Svetozar	Bulgarian Academy of Sciences, Sofia
Marowka, Ami	Bar-Ilan University, Israel
Meyer, Norbert	PSNC, Poznań, Poland
Nabrzyski, Jarek	University of Notre Dame, USA
Oksa, Gabriel	Slovak Academy of Sciences, Bratislava
Olas, Tomasz	Czestochowa University of Technology, Poland
Paprzycki, Marcin	WSM & IBS PAN, Warsaw, Poland
Petcu, Dana	West University of Timisoara, Romania
Quintana-Ortí, Enrique S.	Universitat Jaime I, Spain
Robert, Yves	Ecole Normale Superieure de Lyon, France
Rokicki, Jacek	Warsaw University of Technology, Poland
Rutkowski, Leszek	Częstochowa University of Technology, Poland
Seredyński, Franciszek	Polish Academy of Sciences and Polish-Japanese Institute of Information Technology, Warsaw, Poland
Schaefer, Robert	Institute of Computer Science, AGH, Poland
Silc, Jurij	Jozef Stefan Institute, Slovenia
Sloot, Peter M.A.	University of Amsterdam, The Netherlands
Sosonkina, Masha	Ames Laboratory and Iowa State University, USA
Sousa, Leonel	Technical University of Lisbon, Portugal
Stroiński, Maciej	PSNC, Poznań, Poland
Talia, Domenico	University of Calabria, Italy
Tchernykh, Andrei	CICESE, Ensenada, Mexico

Trinitis, Carsten	TU München, Germany
Trobec, Roman	Jozef Stefan Institute, Slovenia
Trystram, Denis	ID-IMAG, Grenoble, France
Tudruj, Marek	Polish Academy of Sciences and Polish-Japanese Institute of Information Technology, Warsaw, Poland
Tvrdik, Pavel	Czech Technical University, Prague
Vajtersic, Marian	Salzburg University, Austria
Volkert, Jens	Johannes Kepler University, Linz, Austria
Waśniewski, Jerzy	Technical University of Denmark
Wiszniewski, Bogdan	Gdańsk University of Technology, Poland
Yahyapour, Ramin	University of Dortmund, Germany
Zhu, Jianping	University of Texas at Arlington, USA

Table of Contents – Part II

Workshop on Scheduling for Parallel Computing (SPC 2011)

The 4th Workshop on Language-Based Parallel Programming Models (WLPP 2011)

The Second Workshop on Scalable Computing in Distributed Systems and the 7th Workshop on Large Scale Computations on Grids (ScoDiS-LaSCoG 2011)

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

Workshop on Parallel Computational Biology (PBC 2011)

Minisymposium on Applications of Parallel Computation in Industry and Engineering

Minisymposium on High Performance Computing Interval Methods

Workshop on Complex Collective Systems

The FirstWorkshop on Service Oriented Architecture in Distributed Systems (SOADS 2011)

Table of Contents – Part I

Parallel Numerics

Parallel Non-numerical Algorithms

Tools and Environments for Parallel/Distributed/Grid Computing

Applications of Parallel/Distributed Computing

Applied Mathematics, Neural Networks and Evolutionary Computing

Minisymposium on GPU Computing

Workshop on Memory and Data Parallelism on Multi- and Manycore Platforms

Workshop on Models, Algorithms and Methodologies for Hierarchical Parallelism in New HPC Systems

Parallel Cost Function Determination on GPU for the Job Shop Scheduling Problem

Wojciech Bożejko[1], Mariusz Uchroński[1,2], and Mieczysław Wodecki[3]

[1] Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl
[2] Wrocław Centre of Networking and Supercomputing
Wyb. Wyspańskiego 27, 50-370 Wrocław, Poland
mariusz.uchronski@pwr.wroc.pl
[3] Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland
mwd@ii.uni.wroc.pl

Abstract. The goal of this paper is to propose a methodology of the effective cost function determination for the job shop scheduling problem in parallel computing environment. Parallel Random Access Machine (PRAM) model is applied for the theoretical analysis of algorithm efficiency. The methods need a fine-grained parallelization, therefore the approach proposed is especially devoted to parallel computing systems with fast shared memory. The methods proposed are tested with CUDA and OpenCL and ran on NVidia and ATI GPUs.

1 Introduction

In this work we are showing the method of parallelization of the job shop problem solving algorithm for GPGPU, consisting in parallelization of the cost function calculations. There are only few papers dealing with single-walk parallel algorithms for the job shop scheduling problem. Most of papers examine multiple-walk parallelization, e.g. parallel metaheuristics, without using of any theoretical properties of this scheduling problem. From the single-walk approaches, Bożejko et al. [2] proposed a simulated annealing metaheuristic for the job shop problem. Steinhöfel et al. [6] described the method of parallel cost function determination in $O(\log^2 o)$ time on $O(o^3)$ processors, where o is the number of all operations. Bożejko [3] considered a method of parallel cost function calculation for the flow shop problem which constitutes a special case of the job shop problem. Here we show a cost-optimal parallelization which takes a $O(d)$ time on $O(o/d)$ processors, where d is the number of layers in the topological sorted graph representing a solution. Finally, we conduct computational experiments on two types of GPU architectures (provided by nVidia and ATI) which fully confirm theoretical results.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 1–10, 2012.

2 The Job Shop Problem

Let us consider a set of jobs $\mathcal{J} = \{1,2,...,n\}$, set of machines $M = \{1,2,...,m\}$ and a set of operations $\mathcal{O} = \{1,2,...,o\}$. A set $\mathcal{O}$ is decomposed into subsets connected with jobs. A job j consists of a sequence o_j operations indexed consecutively by $(l_{j-1}+1, l_{j-1}+2,...l_j)$ which have to be executed in the order, where $l_j = \sum_{i=1}^{j} o_i$ is a total number of operations of the first j jobs, $j = 1,2,...,n$, $l_0 = 0$, $\sum_{i=1}^{n} o_i = o$. An operation i has to be executed on the machine $v_i \in M$ without any idleness in the $p_i > 0$, $i \in \mathcal{O}$ time. Each machine can execute at most one operation in any moment of time. A feasible solution constitutes a vector of times of the operation execution beginning $S = (S_1, S_2,...,S_o)$ such that the following constrains are fulfilled:

$$S_{l_{j-1}+1} \geq 0, \quad j = 1, 2, ..., n, \tag{1}$$

$$S_i + p_i \leq S_{i+1}, \quad i = l_{j-1} + 1, l_{j-1} + 2, ..., l_j - 1, \quad j = 1, 2, ..., n, \tag{2}$$

$$S_i + p_i \leq S_j \quad \text{or} \quad S_j + p_j \leq S_i, \quad i, j \in \mathcal{O}, \quad v_i = v_j, \quad i \neq j. \tag{3}$$

Certainly, $C_j = S_j + p_j$. An appropriate criterion function has to be added to the above constrains. The most frequently met are the following two criteria: minimization of the time of finishing of all the jobs and minimization of the sum of jobs' finishing times. From the formulation of the problem we have $\mathcal{C}_j \equiv C_{l_j}$, $j \in \mathcal{J}$.

The first criterion, the time of finishing of all the jobs:

$$C_{\max}(S) = \max_{1 \leq j \leq n} C_{l_j}, \tag{4}$$

corresponds to the problem denoted as $J||C_{max}$ in the literature. The second criterion, the sum of the jobs' finishing times:

$$C(S) = \sum_{j=1}^{n} C_{l_j}, \tag{5}$$

corresponds to the problem denoted as $J||\sum C_i$ in the literature.

Both described problems are strongly NP-hard and although they are similarly modeled, the second one is considered to be harder because of lack of some specific properties (so-called block properties, see [5]). They are used in optimization of execution time of solving algorithms.

2.1 Disjunctive Model

A disjunctive model is based on the notion of disjunctive graph $G^* = (\mathcal{O}^*, U^* \cup V)$. This graph has a set of vertices $\mathcal{O}^* = \mathcal{O} \cup \{0\}$ which represents operations (with an additional artificial beginning operation (0), for which $p_0 = 0$), a set

of conjunctive arcs (directed) which show a technological order of operation's execution

$$U^* = U \cup U^0 = \bigcup_{j=1}^{n} \bigcup_{i=l_{j-1}+1}^{l_j-1} \{(i, i+1)\} \cup \bigcup_{j=1}^{n} \{(0, l_{j-1}+1)\} \tag{6}$$

and the set of disjunctive arcs (non-directed) which shows a possible schedule of operations' realization on each machine

$$V = \bigcup_{i,j \in \mathcal{O}, i \neq j, v_i = v_j} \{(i,j),(j,i)\}. \tag{7}$$

Disjunctive arcs $\{(i,j),(j,i)\}$ are in fact pairs of directed arcs with inverted directions connecting vertices i and j.

A vertex $i \in \mathcal{O}$ has a weight p_i which equals to the time of execution of the operation O_i. Arcs have the weight zero. A choice of exactly one arc from the set $\{(i,j),(j,i)\}$ corresponds to determination of a schedule of operations execution – "i before j" or "j before i". A subset $W \subset V$ consisting exclusively of directed arcs, at most one from each pair $\{(i,j),(j,i)\}$, we call a *representation* of disjunctive arcs. Such a representation is complete, if all the disjunctive arcs have determined directions. A complete representation, defining a precedence relation of jobs' execution on the same machine, generates one solution – not always feasible, if it includes cycles. A feasible solution is generated by a complete representation W such that the graph $G(W) = (\mathcal{O}, U \cup W)$ is acyclic. For a feasible schedule values S_i of the vector of operations execution starting times S= (S_1, S_2,...,S_o) can be determined as a length of the longest path incoming to the vertex i (without p_i). Because the graph $G(W)$ includes o vertices and $O(o^2)$ arcs, therefore determining the value of the cost function for a given representation W takes the $O(o^2)$ time.

2.2 Combinatorial Model

In case of many applications a combinatorial representation of the solution is better than a disjunctive model for the job shop problem. It is voided of redundance, characteristic for the disjunctive graph, it denotes the situation where many disjunctive graphs represent the same solution of the job shop problem. A set of operations $\mathcal{O}$ can be decomposed into subsets of operations executed on the single, determined machine $k \in M$, $M_k = \{i \in \mathcal{O} : v_i = k\}$ and let $m_k = |M_k|$. A schedule of operations execution on a machine k is determined by a permutation $\pi_k = (\pi_k(1), \pi_k(2), \ldots, \pi_k(m_k))$ of elements of the set M_k, $k \in M$, where $\pi_k(i)$ means such an element from M_k which is on the i position in π_k. Let $\Pi(M_k)$ be a set of all permutations of elements of M_k. A schedule of operations' execution on all machines is defined as $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, where $\pi \in \Pi$, $\Pi = \Pi(M_1) \times \Pi(M_2) \times \ldots \times \Pi(M_m)$. For a schedule π we create a directed graph (digraph) $G(\pi) = (\mathcal{O}, U \cup E(\pi))$ with a set of vertices $\mathcal{O}$ and a set of arcs $U \cup E(\pi)$), where U is a set of constant arcs representing technological

order of operations execution inside a job and a set of arcs representing an order of operations' execution on machines is defined as

$$E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k-1} \{(\pi_k(i), \pi_k(i+1))\} \tag{8}$$

Each vertex $i \in \mathcal{O}$ has the weight p_i, each arc has the weight zero. A schedule π is feasible, if the graph $G(\pi)$ does not include a cycle. For a given feasible schedule π the process of determining of the cost function value requires the $O(o)$ time, thus shorter than for the disjunctive representation.

3 Sequential Determination of the Cost Function

Taking into consideration the constraints (1)–(3) presented in Section 2 it is possible to determine the time moments of operation completion C_j, $j \in \mathcal{O}$ and job beginning S_j, $j \in \mathcal{O}$ in time $O(o)$ on the sequential machine using the recurrent formula

$$S_j = \max\{S_i + p_i, S_k + p_k\}, j \in \mathcal{O}. \tag{9}$$

where an operation i is a direct technological predecessor of the operation $j \in \mathcal{O}$ and an operation k is a directed machine predecessor of the operation $j \in \mathcal{O}$. The determination procedure of $S_j, j \in \mathcal{O}$ from the recurrent formula (9) should be initiated by an assignment $S_j = 0$ for those operations j which do not possess any technological or machine predecessors. Next, in each iteration an operation j has to be chosen for which:

1. the execution beginning moment S_j has not been determined yet, and
2. these moments were determined for all its direct technological and machine predecessors; for such an operation j the execution beginning moment can be determined from (9).

It is easy to observe that the order of determining S_j times corresponds to the index of the vertex of the $G(\pi)$ graph connected with a j operation after the topological sorting of this graph. The method mentioned above is in fact a simplistic sequential topological sort algorithm without indexing of operations (vertices of the graph). If we add an element of indexing vertices to this algorithm, for which we calculate S_j value, we obtain a sequence which is the topological order of vertices of the graph $G(\pi)$. Now, we define *layers* of the graph collecting vertices (i.e., operations) for which we can calculate S_j in parallel, as we have calculated the starting times for all machines and technological predecessors of the operations in the layer.

Definition 1. *The layer of the graph $G(\pi)$ is a maximal (due to the number of vertices) subsequence of the sequence of vertices ordered by the topological sort algorithm, such that there are no arcs between vertices of this subsequence.*

We will need this definition in the next paragraph.

4 Parallel Determination of the Cost Function

Theorem 1. *For a fixed feasible operations π order for the $J||C_{\max}$ problem, the number of layers from Definition 1 of the $G(\pi)$ graph can be calculated in $O(\log^2 o)$ time on the CREW PRAMs with $O\left(\frac{o^3}{\log o}\right)$ processors.*

Proof. Here we use the $G^*(\pi)$ graph with an additional vertex 0. Let $B = [b_{ij}]$ be an incidence matrix for the $G^*(\pi)$ graph, i.e., $b_{ij} = 1$, if there is an arc i, j in the $G^*(\pi)$ graph, otherwise $b_{ij} = 0$, $i, j = 0, 1, 2, \ldots, o$. The proof is given in three steps.

1. Let us calculate the longest paths (in the sense of the number of vertices) in $G^*(\pi)$. We can use the parallel Bellman-Ford algorithm (see [1]) – we need the time $O(\log^2 o)$ and CREW PRAMs with $O(o^3/\log o)$ processors.
2. We sort distances from the 0 vertex to each vertex in an increasing order. Their indexes, after having been sorted, correspond to the topological order of vertices. This takes the time $O(\log o)$ and CREW PRAMs with $o + 1 = O(o)$ processors, using the parallel mergesort algorithm. We obtain a sequence $Topo[i]$, $i = 0, 1, 2, \ldots, o$. The value of $Topo[o]$ equals d.

We can also use a tree-based parallel maximum determination algorithm in Step 2, instead of mergesort. However, the most time- and processor-consuming is Step 1. We need the time $O(\log^2 o)$ and the number of processors $O\left(\frac{o^3}{\log o}\right)$ of the CREW PRAMs. ∎

Theorem 2. *For a fixed feasible operations π order for the $J||C_{\max}$ problem, the value of cost function can be determined in $O(d)$ time on $O(o/d)$-processor CREW PRAMs, where d is the number of layers of the graph $G(\pi)$.*

Proof. Let Γ_k, $k = 1, 2, \ldots, d$, be the number of calculations of the operations finishing moments C_i, $i = 1, 2, \ldots, o$ in the k-th layer. Certainly $\sum_{k=1}^{d} \Gamma_k = o$. Let p be the number of processors used. The time of computations in a single layer k after having divided calculations into $\lceil \frac{\Gamma_k}{p} \rceil$ groups, each group containing (at most) p elements, is $\lceil \frac{\Gamma_k}{p} \rceil$ (the last group cannot be full). Therefore, the total computation time in all d layers equals $\sum_{k=1}^{d} \lceil \frac{\Gamma_k}{p} \rceil \leq \sum_{k=1}^{d} (\frac{\Gamma_k}{p} + 1) = \frac{o}{p} + d$. To obtain the time of computations $O(d)$we should use $p = O(\frac{o}{d})$ processors. ∎

This theorem provides a cost-optimal method of parallel calculation of the cost function value for the job shop problem with the makespan criterion.

5 The GPU Algorithm

The main part of our parallel implementation of goal function calculation for the job shop problem constitutes calculating of the longest path between all vertices in graph. This part was parallelized with CUDA and OpenCL and ran on NVidia and ATI GPUs.

```
__global__ void PathsKernel(const int o, int *graph, const int i)
{
  int x = threadIdx.x;
  int y = blockIdx.x;
  int k = i;
  int yXwidth = y * (o+1);

  int dYtoX = graph[yXwidth + x];
  int dYtoK = graph[yXwidth + k];
  int dKtoX = graph[k*(o+1) + x];

  int indirectDistance = dYtoK + dKtoX;
  int max = 0;
  int tmp = 0;

  if(dYtoK !=0 and dKtoX !=0)
  {
    tmp = indirectDistance;
    if(max < tmp)
      max = tmp;
  }
  if(dYtoX < max)
  {
    graph[yXwidth + x] = max;
  }
}
```

Fig. 1. CUDA kernel code

Kernel code (Figure 1 – CUDA kernel code) is invoked o times where o is the number of nodes in the graph. At the k-th iteration, the kernel computes direct and the indirect distance between every pair of nodes in graph through node v_k. The larger of the two distances is written back to the distance matrix. The final distance matrix reflects the lengths of the longest paths between each pair of nodes in the graph. The inputs of the GPU kernel are the number of the graph nodes, the graph distance matrix and the iteration number. Figure 2 shows OpenCL implementation of computing the longest path between each pair of nodes in a graph.

6 Computational Experiments

The proposed parallel algorithm of goal function calculation for the job shop problem was coded in CUDA and OpenCL and tested on GPU servers in the Wroclaw Centre for Networking and Supercomputing. The algorithm was tested on the set of benchmark problem instances taken from Lawrence [4] and Taillard [7]. We run our algorithm on three different GPUs:

- NVidia GTX480 with 480 parallel processor cores and 1.4 GHz clock rate,
- ATI Radeon HD5870 with 20 compute units and 850 MHz clock rate,
- ATI Radeon HD5970 with 20 compute units and 725 MHz clock rate.

```
// create the device memory for graph
cl_mem clGraph = clCreateBuffer(context, CL_MEM_READ_WRITE,
                                dataSize, NULL, NULL);
// transfer the input data into device memory
clEnqueueWriteBuffer(commands, clGraph,
                     CL_TRUE, 0, dataSize,
                     graph, 0, NULL, NULL);
size_t local  = o+1;
size_t global = (o+1)*(o+1);
// set the arguments to the compute kernel
clSetKernelArg(kernel, 0, sizeof(int), &o);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &clGraph);
for (int iter=1; iter <= size+1; iter++)
{
  for(int i=0; i<=o; ++i)
  {
    clSetKernelArg(kernel, 2, sizeof(int), &i);
    // execute the kernel
    clEnqueueNDRangeKernel(commands, kernel, 1, NULL,
                           &global, &local, 0, NULL, NULL);
    // wait for all commands to complete
    clFinish(commands);
  }
}
// read back the results from the device
clEnqueueReadBuffer(commands, clGraph, CL_TRUE, 0, dataSize,
                    graph, 0, NULL, NULL );
```

Fig. 2. OpenCL code

This GPUs are installed in servers with Intel Core i7 CPU with 3.20 GHz clock rate working under 64-bit GNU/Linux Ubuntu 10.10 operating system. The proposed algorithm uses o^2 processors for layers determination (basing on Theorem 1, scaled to the time $O(o \log o)$). The sequential algorithm (based on the method presented in the Section 3) using one CPU processor has been coded with the aim of determining the *absolute* speedup value which can be obtained with a parallel algorithm.

Our CUDA (OpenCL) implementation of the parallel goal function calculation for the job shop problem uses o blocks (work groups) with o threads (work items). The maximum work item size per dimension for HD5870 and HD5970 is equal to 256. Therefore we could ran our parallel algorithm on ATI GPUs only for Lawrence test instances (up to 255 operations). The maximum number of threads per block for NVidia GTX480 GPU is equal to 1024. On this GPU we can calculate a goal function for the job shop problem with up to 1024 operations.

Figures 3 and 4 show the comparison of computation time for sequential (run on CPU) and parallel algorithm coded in CUDA/OpenCL and run on NVidia/ATI GPUs. The measured time contains the time needed for data transfer between CPU and GPU. As shown in Figure 3 the considered algorithm coded in CUDA is faster than the algorithm coded in OpenCL for all Lawrence test instances. The algorithm coded in OpenCL is faster than the algorithm for CPU

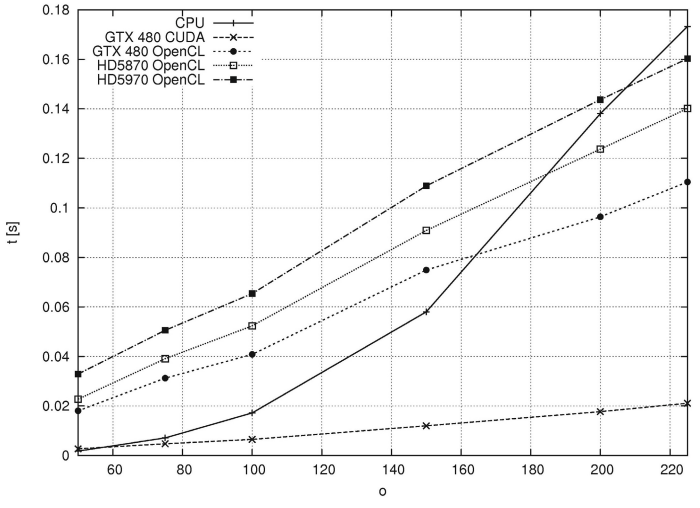


Fig. 3. Computation time for Lawrence test instances

for the number of operations grater than 225. The comparison of computation time for OpenCL code on different hardware shows that running the same code on Nvidia GTX 480 take less time than on used ATI GPUs. Also, the computation time for ATI Radeon HD5870 is less than ATI Radeon HD5970. This situation is caused by clock rate for GPU hardware used for tests evaluation.

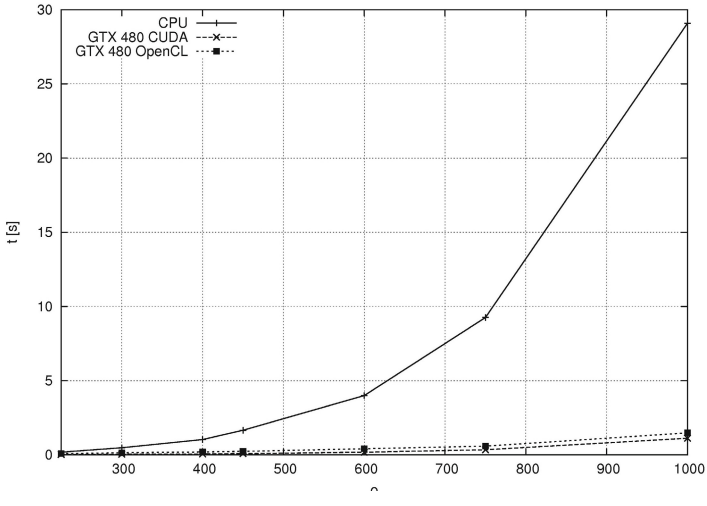


Fig. 4. Computation time for Taillard test instances

Figure 5 and Table 1 report the comparison of speedup obtained for CUDA and OpenCL implementation on NVidia GTX480 GPU. The particular columns in Table 1 denote:

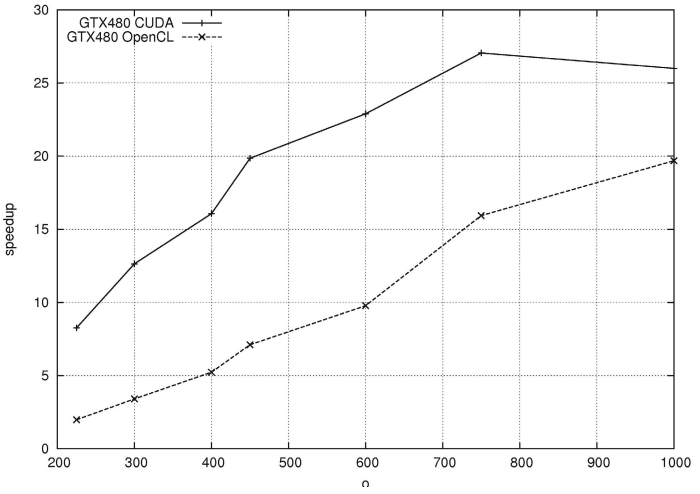


Fig. 5. Speedup for Taillard test instances

Table 1. Speedup for CUDA and OpenCL impementation obtained on NVidia GTX480 GPU

problem	o	s_{OpenCL}	s_{CUDA}
tail01-10	225	1.97731	8.25768
tail11-20	300	3.40649	12.6401
tail21-30	400	5.22559	16.0617
tail31-40	450	7.10766	19.855
tail41-50	600	9.7754	22.8953
tail51-60	750	15.9346	27.0535
tail61-70	1000	19.6828	25.9972

- o – number of operations in considered job shop problem instance,
- s_{OpenCL} – speedup for OpenCL implementation (average for each 10 instances),
- s_{CUDA} – speedup for CUDA implementation (average for each 10 instances).

The obtained results show that parallel goal function computation for the job shop problem on GPU results in shorter calculation time for the number of operations greater than 225. The considered algorithm coded in CUDA allow to obtain greater speedup than for the algorithm coded in OpenCL for all tested benchmarks. Our parallel algorithm reached 25x speedup for CUDA implementation and 19x speedup for OpenCL implementation on NVidia GTX480 GPU.

7 Conclusions

The algorithm proposed in this paper can be used for computation acceleration in metaheurisics solving the job shop problem. The calculation time of goal function in algorithms which solve the job shop problem take even 90% of the whole algorithm computation time. The use of parallel algorithm for goal function calculation might result in significant decreasing of algorithm execution time for solving the job shop problem.

References

1. Bożejko, W.: A new class of parallel scheduling algorithms. Monographs series. Wroclaw University of Technology Publishing House (2010)
2. Bożejko, W., Pempera, J., Smutnicki, C.: Parallel Simulated Annealing for the Job Shop Scheduling Problem. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009, Part I. LNCS, vol. 5544, pp. 631–640. Springer, Heidelberg (2009)
3. Bożejko, W.: Solving the flow shop problem by parallel programming. Journal of Parallel and Distributed Computing 69, 470–481 (2009)
4. Lawrence, S.: Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Graduate school of industrial administration. Carnegie Mellon University, Pittsburgh (1984)
5. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the job shop problem. Management Science 42, 797–813 (1996)
6. Steinhöfel, K., Albrecht, A., Wong, C.K.: Fast parallel heuristics for the job shop scheduling problem. Computers & Operations Research 29, 151–169 (2002)
7. Taillard, E.D.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)

Partitioning and Scheduling Workflows across Multiple Sites with Storage Constraints

Weiwei Chen and Ewa Deelman

Information Sciences Insitute,
University of Southern California, Marina del Rey, CA, 90292, USA
{wchen,deelman}@isi.edu

Abstract. This paper aims to address the problem of scheduling large workflows onto multiple execution sites with storage constraints. Three heuristics are proposed to first partition the workflow into sub-workflows. Three estimators and two schedulers are then used to schedule sub-workflows to the execution sites. Performance with three real-world workflows shows that this approach is able to satisfy storage constraints and improve the overall runtime by up to 48% over a default whole-workflow scheduling.

Keywords: workflow scheduling, partitioning, storage constraints.

1 Introduction

Scientific workflows [1] have been widely applied in astronomy [2], seismology [3], genomics [4], etc. A scientific workflow has a sequence of jobs that perform required functionality and it has control or data dependencies between jobs. Workflows in systems such as Pegasus [5] are defined at an abstract level, devoid of resource assignments. A key problem that needs to be addressed is the mapping of jobs in the workflow onto resources that are distributed in the wide area. This is especially challenging for data-intensive workflows that require significant amount of storage. For these workflows, we need to use multiple execution sites and consider their available storage. For example, the CyberShake [3] workflow has 80 sub-workflows and each sub-workflow has more than 24,000 individual jobs and 58 GB data. A sub-workflow is a workflow and also a job of a higher-level workflow. We use Condor [6] pools as execution sites.

In this paper, we have developed a three-phase scheduling approach integrated with the Pegasus Workflow Management System [5] to partition, estimate, and schedule workflows onto distributed resources. Pegasus is a workflow-mapping and execution engine that is used to map large-scale scientific workflows onto the available resources. Our contributions include three heuristics to partition workflows respecting storage constraints and internal job parallelism. We utilize three methods to estimate runtime of sub-workflows and then we schedule them based on two commonly used algorithms (MinMin[7] and HEFT[8]).

The reason that we partition workflows into sub-workflows instead of scheduling individual jobs is that this approach reduces the complexity of the workflow

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 11–20, 2012.

mapping. For example, the entire CyberShake workflow has more than 1.9×10^5 tasks, which is a large number for workflow management tools. In contrast, each sub-workflow has 24,000 tasks, which is acceptable for these tools.

We model workflows as Directed Acyclic Graphs (DAGs), where nodes represent computation and directed edges represent data flow dependencies between nodes. Such workflows comprise sequences of fan-out (where the output of a job is input to many children), fan-in (where the output of several jobs is aggregated by a child), and pipeline nodes (1 parent, 1 child). We assume that the size of each input file and output file is known, and that they are much smaller than the storage constraints of a site. To estimate the runtime of sub-workflows, runtime information for each job is required. In our proposed heuristics, we use historical performance information.

2 Related Work

Considerable amount of work tried to solve workflow-mapping problem using DAG scheduling heuristics such as HEFT[8], Min-Min[7], etc. Sonmez [10] extended them to multiple workflows in multi-cluster grids. Duan [9], Wieczorek [16] have discussed the scheduling and partitioning of scientific workflows in dynamic grids. These algorithms do not take storage constraints into consideration and they need to check every job and schedule it, while our algorithm only needs to check a few particular types of jobs (see Section 3).

Singh[11] optimized disk usage and runtime performance by removing data files when they're no longer required. We distinguish our work in three points. First, there is an upper bound of the amount of data that can be cleaned up. If the workflow with data clean up is still too large for a single site to run, our work tries to find a valid partitioning if it exists. Second, our algorithm only needs to check a few particular jobs instead of the entire workflow. Third, simply applying scheduling algorithms to this problem and grouping jobs at the same sites into sub-workflows may result in invalid workflows with cross dependencies (see Section 3). Data clean up can be simply added to our approach.

Workflow partitioning can be classified as a network cut problem [12] where a sub-workflow is viewed as a sub-graph. But there are two differences with our approach. First, we must consider the problem of data overlap when a new job is added to a sub-workflow. Second, valid workflows require no cross dependencies although it is possible to make that cut in network cut problem.

3 System Design

Our approach (shown in Fig.1) has three phases: partition, estimate and schedule. The partitioner takes the original workflow and site catalog (containing information about available execution sites) [5] as input, and outputs various sub-workflows that respect the storage constraints—this means that the data requirements of a sub-workflow are within the data storage limit of a site. The site catalog provides information about the available resources. The estimator

provides the runtime estimation of the sub-workflows and supports three estimation methods. The scheduler maps these sub-workflows to resources considering storage requirement and runtime estimation. The scheduler supports two commonly used algorithms. We first try to find a valid mapping of sub-workflows satisfying storage constraints. Then we optimize performance based on these generated sub-workflows and schedule them to appropriate execution sites if runtime information for individual jobs is already known. If not, a static scheduler maps them to resources merely based on storage requirements.

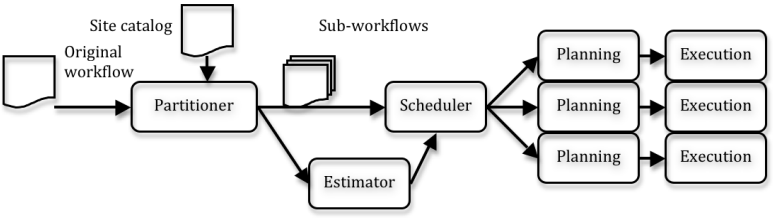


Fig. 1. The steps to partition and schedule a workflow

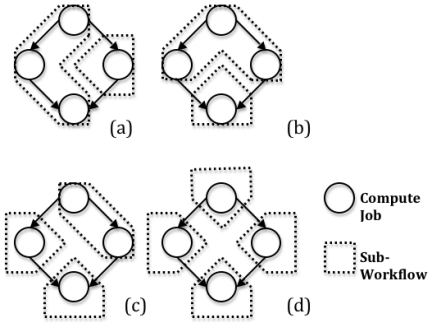


Fig. 2. Four Partitioning

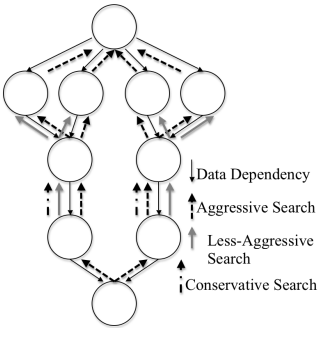


Fig. 3. Three Steps of Searches

Cross Dependency. The major challenge in partitioning workflows is to avoid cross dependency, which is a chain of dependencies that forms a cycle in the graph (in this case cycles between sub-workflows). With cross dependencies, workflows are not able to proceed since they form a deadlock loop. For a workflow depicted in Fig.2, we show the result of four different partitioning. Partitioning (a) does not work in practice since it has a deadlock loop. Partitioning (c) is valid but not efficient compared to Partitioning (b) or (d) that have more parallelism.

Partitioner. Usually jobs that have parent-child relationships share a lot of data since they have data dependencies. It is reasonable to schedule such jobs into the same partition to avoid extra data transfer and also to reduce the overall runtime. Thus, we propose Heuristic I to find a group of parents and

children. Our heuristic only checks three particular types of nodes: the fan-out job, the fan-in job, and the parents of the fan-in job and search for the potential candidate jobs that have parent-child relationships between them. The check operation means checking whether one particular job and its potential candidate jobs can be added to a sub-workflow while respecting storage constraints. Thus, our algorithm reduces the time complexity of check operations by n folds, while n is the average depth of the fan-in-fan-out structure. The check operation takes more time than the search operation since the calculation of data usage needs to check all the data allocated to a site and see if there is data overlap. Similar to [8], the algorithm starts from the sink job and proceeds upward.

```
Input: Workflow G
Input: Site List SL[index], which stores all
information about a compute site.
Output: Sub-workflow List SWL[index], which has
all the sub-workflows scheduled to a compute site.
Let index = 0;
Q = new Queue()
Add the sink job of G to Q
S = new subworkflow()
While Q is not empty
  Let j be the last job in Q
  //for fan-in job
  Do Aggressive-Search from j
  Let C be the list of potential candidate jobs to be
added to S on SL[index]
  Let P be the list of parents of all candidates
  Let D be the data size on SL[index] with C
  If(D > storage constraint of SL[index])
  Do Less-Aggressive-Search from j, update C, P, D
  If(D > storage constraint of SL[index])
    Do Conservative-Search from j, update C, P, D
  EndIf
  EndIf
  // for other types of job
  ...
  If(S causes cross dependency at SL[index])
    S = new subworkflow()
  EndIf
  Add all the jobs in C to S
  Add all the jobs in P to the head of Q
  add S to SWL[index]
  If(SL[index] has no enough space left)
    index ++
  EndIf
  //for other situations
  ...
  Remove j from Q
EndWhile
Return SWL
```

Fig. 4. Pseudo-code of partitioning. Not all the situations are listed here.

To search for the potential candidate jobs that have parent-child relationships, the partitioner tries three steps of searches. For a fan-in job, it first checks if it is possible to add the whole fan structure into the sub-workflow (aggressive search). If not, similar to Fig.2(d), a cut is issued between this fan-in job and its parents to avoid cross dependencies and increase parallelism. Then a less aggressive search is performed on its parent jobs, which includes all of its predecessors until the search reaches a fan-out job. If the partition is still too large, a conservative search is performed, which includes all of its predecessors until the search reaches a fan-in job or a fan-out job. Fig.3 depicts an example of three steps of search while the workflow in it has an average depth of 4. Pseudo-code of Heuristic I is depicted in Fig.4.

The partitioner starts by picking an execution site from site catalog and forming a sub-workflow with the heuristic above. Users can specify the order of execution sites to be picked or the partitioner will sort them in the order of storage constraints. If the execution site does not have sufficient storage to host any more jobs, a new execution site is selected. For the dependencies between jobs across multiple sub-workflows, they form the new dependencies between sub-workflows

and are added to the final graph. The partitioner guarantees to satisfy storage constraints since in each step it assures the size of all sub-workflows assigned to a site is smaller than its storage constraint.

To compare the approach we propose two other heuristics. The motivation for Heuristic II is that Partitioning (c) in Fig.2 is able to solve the problem. The motivation for Heuristic III is an observation that partitioning a fan structure into multiple horizontal levels is able to solve the problem. **Heuristic II** adds a job to a sub-workflow if all of its unscheduled children can be added to that sub-workflow without causing cross dependencies or exceed the storage constraint. **Heuristic III** adds a job to a sub-workflow if two conditions are met: 1) for a job with multiple children, each child has already been scheduled; 2) after adding this job to the sub-workflow, the data size does not exceed the storage constraint.

Estimator. To optimize the workflow performance, runtime estimation for sub-workflows is required assuming runtime information for each job is already known. We provide three methods. **Critical Path** is defined as the longest depth of the sub-workflow weighted by the runtime of each job. **Average CPU Time** is the quotient of cumulative CPU time of all jobs divided by the number of available resources. The **HEFT** estimator uses the calculated earliest finish time of the last sink job as makespan of sub-workflows assuming that we use HEFT to schedule sub-workflows.

Scheduler. The scheduler selects appropriate resources for the sub-workflows satisfying the storage constraints and optimizes the runtime performance. We select **HEFT**[8] and **MinMin**[7]. There are two differences compared to their original versions. First, the data transfer cost within a sub-workflow is ignored since we use a shared file system in our experiments. Second, the data constraints must be satisfied for each sub-workflow. The scheduler selects a near-optimal set of resources in terms of available Condor slots since its the major factor influencing the performance. Although some more comprehensive algorithms can be adopted, HEFT or MinMin are able to improve the performance significantly in terms that the sub-workflows are already generated since the number of sub-workflows has been greatly reduced compared to the number of individual jobs.

4 Experiments and Evaluations

In order to quickly deploy and reconfigure computational resources, we use a cloud computing resource in FutureGrid [17] running Eucalyptus [13]. Eucalyptus is an infrastructure software that provides on-demand access to Virtual Machine (VM) resources. In all the experiments, each VM has 4 CPU cores, 2 Condor slots, 4GB RAM and has a shared file system mounted to make sure data staged into a site is accessible to all compute nodes. In the initial experiments we build up four clusters, each with 4 VMs, 8 Condor slots. In the last experiment of site selection, the four virtual clusters are reconfigured and each cluster has 4, 8, 10 and 10 Condor slots respectively. The submit host that performs workflow planning and which sends jobs to the execution sites is a Linux

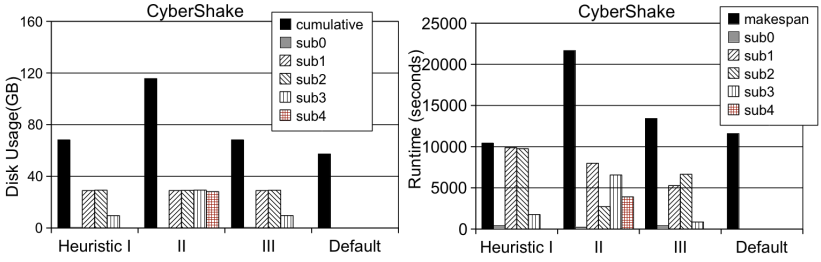


Fig. 5. Performance of the three heuristics. The default workflow has one execution site with 4 VMs and 8 Condor slots and has no storage constraint.

2.6 machine equipped with 8GB RAM and an Intel 2.66GHz Quad CPUs. We use Pegasus to plan the workflows and then submit them to Condor DAGMan [14], which provides the workflow execution engine. Each execution site contains a Condor pool and a head node visible to the network.

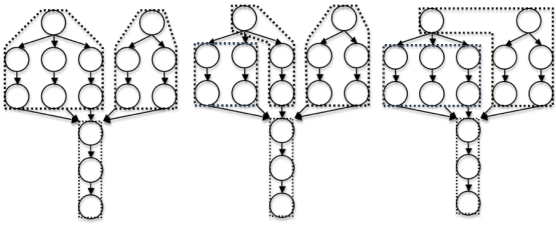
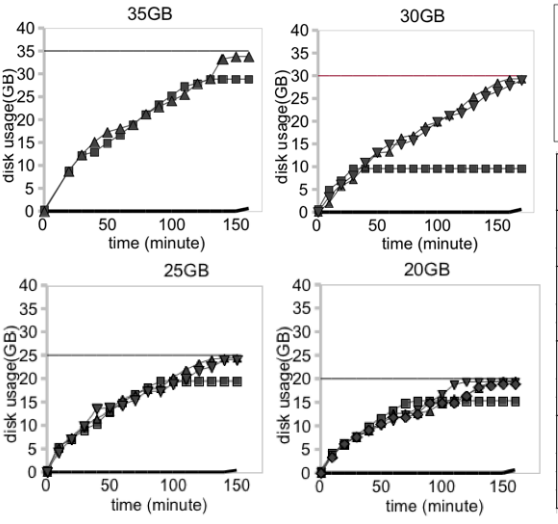


Fig. 6. From left to right: Heuristic I, Heuristic II, Heuristic III

Performance Metrics. To evaluate the performance, we use two types of metrics. **Satisfying the Storage Constraints** is the main goal of our work in order to fit the sub-workflows into the available storage resources. We compare the results of different storage constraints and heuristics. **Improving the Runtime Performance** is the second metric that is concerned with to minimize the overall makespan. We compare the results of different partitioners, estimators and schedulers.

Workflows Used. We ran three different workflow applications: an astronomy application (Montage), a seismology application (CyberShake) and a bioinformatics application (Epigenomics). They were chosen because they represent a wide range of application domains and a variety of resource requirements [15]. For example, Montage is I/O intensive, CyberShake is memory intensive, and Epigenomics is CPU intensive. The goal of the CyberShake Project [3] is to calculate Probabilistic Seismic Hazard curves for locations in Southern California area. We ran one partition that has 24,132 tasks and 58GB of overall data. Montage [2] is an astronomy application that is used to construct large image mosaics



storage	site	disk usage(GB)	percentage
35GB	A	sub0: 0.06 sub1: 33.8	97%
	B	sub2: 28.8	82%
30GB	A	sub0: 0.07 sub1: 29.0	97%
	B	sub2: 29.3	98%
	C	sub3: 28.8	96%
25GB	A	sub0: 0.06 sub1: 24.1	97%
	B	sub2: 24.4	98%
	C	sub3: 19.5	78%
20GB	A	sub0: 0.06 sub1: 18.9	95%
	B	sub2: 19.3	97%
	C	sub3: 19.6	98%
	D	sub4: 15.3	77%

Fig. 7. CyberShake with storage constraints of 35GB, 30GB, 25GB, and 20GB. They have 3, 4, 4, and 5 sub-workflows and require 2, 3, 3, and 4 sites to run respectively.

of the sky. We ran a Montage workflow with a size of 8 degree square of sky. The workflow has 10,422 tasks and 57GB of overall data. Epigenomics [4] maps short DNA segments collected with gene sequencing machines to a reference genome. The workflow has 1,586 tasks and 23GB of overall data. We ran each workflow instance 5 times to assure the variance is within 10%.

Performance of Different Heuristics. We compare the three proposed heuristics with the CyberShake application. The storage constraint for each site is 30GB. Heuristic II produces 5 sub-workflows with 10 dependencies between them. Heuristic I produces 4 sub-workflows and 3 dependencies. Heuristic III produces 4 sub-workflows and 5 dependencies. The results are shown in Fig.5 and Heuristic I performs better in terms of both runtime reduction and disk usage. This is due to the way it handles the cross dependency. Heuristic II or Heuristic III simply adds a job if it does not violate the storage constraints or the cross dependency constraints. Furthermore, Heuristic I puts the entire fan structure into the same sub-workflow if possible and therefore reduces the dependencies between sub-workflows. The entire fan structure is defined as a set of jobs and begins from a fan-out job and merges into a fan-in job. In Fig.6 with a simplified CyberShake workflow, Heuristic I runs two sub-workflows in parallel while the other two have to run them in sequence. From now on, we only use Heuristic I in the partitioner in our experiments.

Performance with Different Storage Constraints. Fig.7 depicts the disk usage of the CyberShake workflows over time with storage constraints of 35GB, 30GB, 25GB, and 20GB. They are chosen because they represent a variety of required execution sites. Fig.8 depicts the performance of both disk usage and runtime. Storage constraints for all of the sub-workflows are satisfied. Among

them sub1, sub2, sub3 (if exists), and sub4 (if exists) are run in parallel and then sub0 aggregates their work. The CyberShake workflow across two sites with a storage constraint of 35GB performs best. The makespan (overall completion time) improves by 18.38% and the cumulative disk usage increases by 9.5% compared to the default workflow without partitioning or storage constraints. The cumulative data usage is increased because some shared data is transferred to multiple sites. Adding more sites does not improve the makespan because they require more data transfer even though the computation part is improved.

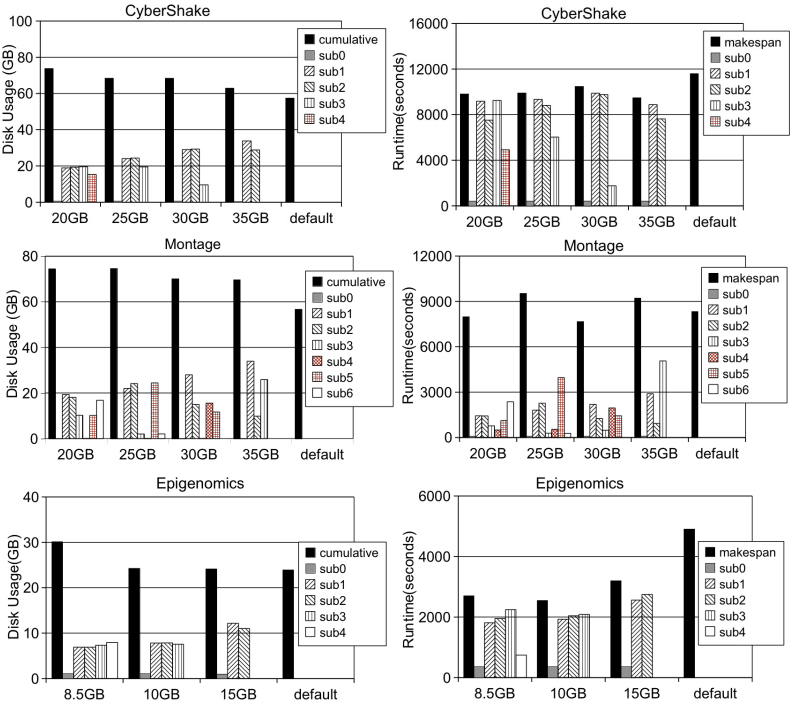
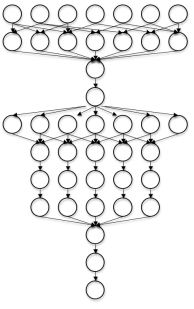


Fig. 8. Performance of CyberShake, Montage and Epigenomics with different storage constraints

Fig.8 depicts the performance of Montage with storage constraints ranging from 20GB to 35GB and Epigenomics with storage constraints ranging from 8.5GB to 15GB. The Montage workflow across 3 sites with 30GB disk space performs best with 8.1% improvement in makespan and the cumulative disk usage increases by 23.5%. The Epigenomics workflow across 3 sites with 10GB storage constraints performs best with 48.1% reduction in makespan and only 1.4% increase in cumulative storage. The reason why Montage performs worse is related to its complex internal structures. Montage has two levels of fan-out-fan-in structures and each level has complex dependencies between them as shown



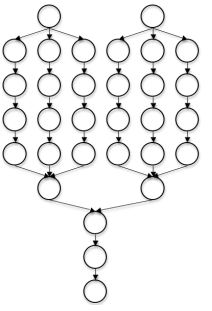
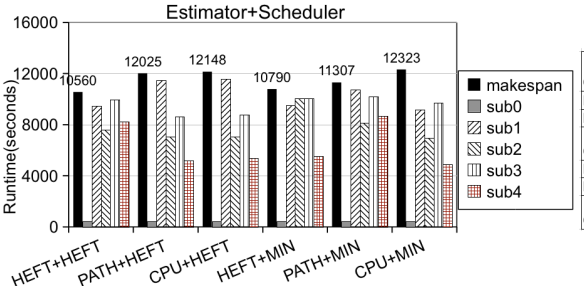


Fig. 9. The Montage (left) and the Epigenomics (right) workflows. For simplicity, only a few branches are shown.

in Fig.9. Our heuristic is not able to untie them thoroughly and thereby the cost of data transfer increases and the sub-workflows are not able to run in parallel.



Combination	Estimator	Scheduler	Makespan (second)
HEFT+HEFT	HEFT	HEFT	10559.5
PATH+HEFT	Critical Path	HEFT	12025.4
CPU+HEFT	Average CPU Time	HEFT	12148.2
HEFT+MIN	HEFT	MinMin	10790
PATH+MIN	Critical Path	MinMin	11307.2
CPU+MIN	Average CPU Time	MinMin	12323.2

Fig. 10. Performance of estimators and schedulers

Site Selection. We use three estimators and two schedulers described in Section 3 together with the CyberShake workflow. We build four execution sites with 4, 8, 10 and 10 Condor slots respectively. The labels in Fig.10 are defined in a way of Estimator+Scheduler. For example, HEFT+HEFT denotes a combination of HEFT estimator and HEFT scheduler, which performs best. The Average CPU Time (or CPU in Fig. 10) does not take the dependencies into consideration and the Critical Path (or PATH in Fig.10) does not consider the resource availability. The HEFT scheduler is slightly better than MinMin scheduler (or MIN in Fig.10). Although HEFT scheduler uses a global optimization algorithm compared to MinMins local optimization, the complexity of scheduling sub-workflows has been greatly reduced compared to scheduling a vast number of individual tasks. Therefore, both of them are able to handle such situations.

5 Conclusions

This paper provides a solution to address the problem of scheduling large workflows across multiple sites with storage constraints. Three heuristics are proposed

and compared to show the close relationship between cross dependency and runtime improvement. The performance with three workflows shows that this approach is able to satisfy the storage constraints and reduce the makespan significantly especially for Epigenomics, which has fewer fan-in (synchronization) jobs.

Acknowledgements. This work is supported by NFS under grant number IIS-0905032, and OCI-0910812 (FutureGrid). We would like to thank G. Juve, K. Vahi and M. A. Amer for their help.

References

1. Taylor, I.J., Deelman, E., et al.: Workflows for e-Science. In: Scientific Workflows for Grids. Springer (2007)
2. Berriman, G.B., et al.: Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In: Proc. of SPIE, vol. 5493, pp. 221–232 (2004)
3. Maechling, P., Deelman, E., et al.: SCEC CyberShake Workflows–Automating Probabilistic Seismic Hazard Analysis Calculations. In: Workflows for e-Science. Scientific Workflows for Grids. Springer (2007)
4. USC Epigenome Center, http://epigenome.usc.edu
5. Deelman, E., et al.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Sci. Program 13, 219–237 (2005)
6. Litzkow, M., et al.: Condor–A Hunter of Idle Workstations. In: ICDCS (June 1988)
7. Blythe, J., et al.: Task Scheduling Strategies for Workflow-Based Applications in Grids. In: CCGrid (2005)
8. Topcuoglu, H., et al.: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE TPDS 13(3) (March 2002)
9. Duan, R., et al.: Run-time Optimisation of Grid Workflow Applications. In: 7th IEEE/ACM Intl. Conf. on Grid Computing (September 2005)
10. Sonmez, O.O.: Application-Oriented Scheduling in Multicluster Grids (June 2010), http://www.pds.ewi.tudelft.nl/~homedirsonmez/research.htm
11. Singh, G., et al.: Optimizing Workflow Data Footprint. Special issue of the Scientific Programming Journal dedicated to Dynamic Computational Workflows: Discovery, Optimisation and Scheduling (2007)
12. Papadimitriou, C.H., et al.: Combinatorial Optimization: Algorithms and Complexity, Dover, pp. 120–128 (1998) ISBN 0486402584
13. Eucalyptus Systems, http://www.eucalyptus.com/
14. Condor Team, http://www.cs.wisc.edu/condor/dagman
15. Juve, G., et al.: Scientific Workflow Applications on Amazon EC2, E-Science Workshops, Oxford UK (December 2009)
16. Wieczorek, M., et al.: Scheduling of scientific workflows in the ASKALON grid environment. SIGMOND Record 34(3) (September 2005)
17. FutureGrid, https://portal.futuregrid.org

Grid Branch-and-Bound for Permutation Flowshop

Maciej Drozdowski, Paweł Marciniak, Grzegorz Pawlak, and Maciej Płaza

Institute of Computing Science, Poznań University of Technology,
Piotrowo 2, 60-965 Poznań, Poland
{Maciej.Drozdowski,Grzegorz.Pawlak}@cs.put.poznan.pl,
Maciej.Plaza@gmail.com

Abstract. Flowshop is an example of a classic hard combinatorial problem. Branch-and-bound is a technique commonly used for solving such hard problems. Together, the two can be used as a benchmark of maturity of parallel processing environment. Grid systems pose a number of hurdles which must be overcome in practical applications. We give a report on applying parallel branch-and-bound for flowshop in grid environment. Methods dealing with the complexities of the environment and the application are proposed, and evaluated.

Keywords: branch-and-bound, flowshop, grid computing.

1 Introduction

Solving a hard combinatorial optimization problem on the grid is considered in this paper. Flowshop is a classic **NP**-hard combinatorial optimization problem. A set of test instances has been proposed for this problem [16]. Despite 20-year attempts, some of the instances remain unsolved. Thus, flowshop became a benchmark problem in combinatorial optimization, and deterministic scheduling.

Though many methods have been proposed to solve combinatorial optimization problems, Branch-and-Bound (BB) remains the main algorithm delivering guaranteed optimum solutions. BB partially enumerates the solutions, and this process often can be envisioned as searching a tree. Various approaches are used to prune the tree, but still search spaces of combinatorial problems remain huge. Therefore, parallel branch-and-bound (PBB) is used to reduce the running time [1,3]. BB parallelization introduces a host of new complications [3,7]. Overcoming them requires making design decisions which influence performance of PBB.

By combining computational resources of many institutions Grid environments provide computational power not available to any single institution separately. Therefore, grid is a very attractive computing platform for combinatorial optimization applications. Yet, grid has a number of inherent peculiarities such as resource heterogeneity and volatility which must be dealt with when designing a robust application.

Overall, the three elements: benchmark hard combinatorial problem, parallel branch-and-bound, the grid environment make the implementation practically

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 21–30, 2012.

hard. Therefore, these three elements can serve as a benchmark of *maturity* and *robustness* in parallel processing. With such a goal 2nd Grid Plugtests Flowshop Challenge was organized by European Telecommunications Standards Institute in 2005 [4]. A testbed for solving benchmark hard problems on the grid was provided to verify usability and maturity of the computational platform and the programming environment. The code presented in this paper scored 1st prize in the above competition. This paper is dedicated to the grid parallel processing, and the challenges which must be faced by application designers, rather than to solving flowshop problem itself. Flowshop serves as a benchmark here.

The rest of this paper is organized as follows. In the next section we define flowshop problem. Section 3 is dedicated to grid middleware, and the computational platform peculiarities. In Section 4 parallel branch-and-bound algorithm is described. In Section 5 we report on the results obtained by our implementation of PBB.

2 Flowshop

Flowshop (FS) problem is defined as follows. Sets $\{M_1, \ldots, M_m\}$ of m dedicated machines, and $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs are given. Each job J_j consists of a sequence of m operations $(O_{j1}, O_{j2}, \ldots, O_{jm})$. Operations O_{ji} are executed on machine M_i, for all $j = 1, \ldots, n$. Consequently, operations $O_{j1}, O_{j2}, \ldots, O_{jm}$ proceed through machines $M_1, M_2, \ldots, M_m$, as for example, cars on the production line. Execution time of operation O_{ji} is a non-negative integer p_{ji}. Only active schedules are allowed, which means that operations are started as soon as it is possible. The problem consists in finding the shortest schedule. We will denote schedule length by C_{max}.

In general, the operations of different jobs can be executed on a given machine in an arbitrary order. This results in at most $(n!)^m$ possible solutions of the problem. In *permutation flowshop* jobs proceed through the machines in the same order. It means that the sequence of operations from different jobs is the same on all the machines. In this paper we consider permutation flowshop. Though there are 'only' $n!$ solutions for the permutation flowshop, the number of possible solutions is still very big in practice. Flowshop problem is polynomially solvable for $m = 2$ [8], and strongly **NP**-hard for $m \geq 3$ [5].

A set of test instances is known [16] for flowshop problem. In the following we refer to Taillard's instances [17] which sizes are from $(n \times m)$: 20×5 to 500×20. Over the years, flowshop has been used as a benchmark problem to test new methods in combinatorial optimization [6,9,14].

3 The Test Platform

In this section we characterize grid environment in the 2nd Grid Plugtests Flowshop Challenge. The information mentioned here follows [4]. Initial runs of the algorithm were also executed on an SMP SunFire 6800 machine with 24 CPUs and 96GB RAM in Poznań Supercomputing and Networking Center.

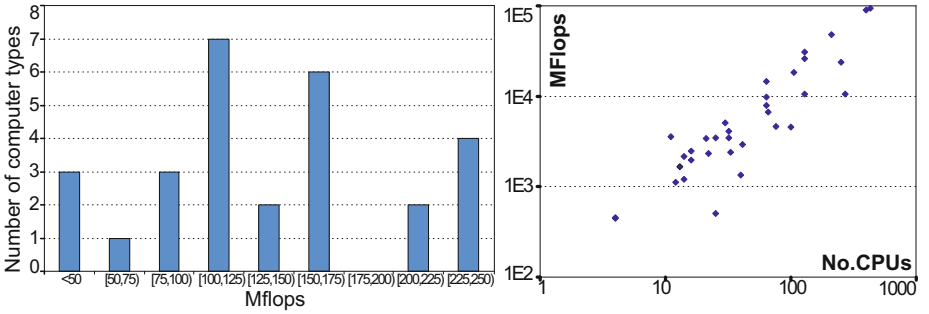


Fig. 1. a) Distribution of computer performance score. b) Datacenter performance vs. CPU number. On the basis of [4].

The test grid consisted of 2700 CPUs distributes in 40 locations (datacenters) in 13 countries on 5 continents [4]. The computing platform comprised 5 different operating systems, 10 job submission and deployment systems, 5 Java Virtual Machine types. The total performance of the grid was estimated at 450GFlops according to SciMark benchmark [4]. Fig.1a depicts distribution of the CPU speeds, and Fig.1b the spread of datacenter CPU numbers and the total performance. It should be noted that the computers were not continuously and exclusively available. Since the computing platform lacked sufficient reliability mechanisms, such mechanisms had to be implemented by the application.

Developing an application for such a diverse computing platform was possible thanks to Proactive middleware [15]. Proactive is a Java programming library providing active objects, asynchronous communication between the objects, their deployment and method execution. An active object has its own thread of execution. For example, it can be executed in a loop actively pooling conditions and reacting to them. Proactive provides uniform view of the application memory space. Therefore, methods of a remote active object can be called by other objects in the same way as in a sequential program. For deployment of the active objects Proactive used XML Deployment Descriptors comprising information on: the addresses of available computers, process initiation, communication and file transfer protocols. Thus, a programmer was separated from the actual grid hardware and referred to an active object in its very own code rather than to a description of a process on a remote computer.

4 Parallel Branch-and-Bound for Grid

In this section we give a general description of the PBB, our implementation of PBB for flowshop, and the above computing platform. An interested reader will find description of BB and PBB in, e.g., [2,3,7,10].

4.1 PBB in General

The BB search for the optimum solution may be understood as a recursive analysis of the set of solutions in divide-and-conquer manner. The initial set of solutions is divided into subsets. A subset is fathomed by either eliminating it, as not containing optimum solution, or by further sub-dividing it. The recursion stops if a subset comprises only one solution. This process of creating and eliminating subsets can be viewed as constructing and searching of a tree. Thus, names sub-tree or search tree node can be used when referring to a subset of solutions. We will use abbreviation *BBTnode* for Branch and Bound tree node.

Search trees for hard combinatorial problems have exponential size in the length of the input (here the number of jobs n). Since it is not desirable, various methods are applied to limit the search tree. We will be calling a BBTnode *active* if it has neither been branched into offspring, nor eliminated. The active nodes can be eliminated on the following basis (e.g.): 1) all the solutions represented by the subtree are infeasible, 2) all the solutions represented by the subtree are dominated, 3) all the solutions represented by the subtree are not better than some already known solution. If it is known that the optimum solution has certain feature, then the analyzed BBTnode a is *dominated* and can be eliminated if it lacks such a feature. Suppose we solve minimization problem. In the third case a lower bound $LB(a)$ on the objective function is calculated for all the solutions represented by a. Suppose some solution b is known with objective value $C_{max}(b) \leq LB(a)$, then node a is pruned. The best known solution b establishes an upper bound UB. The more the search tree is cut, the smaller the set of visited nodes and the faster the algorithm is. Yet, sometimes shallower cuts but faster to calculate give faster BB than deep time consuming cuts.

Parallelization is a natural step to speed up BB. Yet, PBB has to deal with several problems. A common approach is to distribute the search tree between computers. Since the search tree is instance-dependent, its structure is unknown before the runtime. Consequently, the tree cannot be partitioned statically because some computers would quickly run out of work, while the other would be overloaded. However, an optimistic scenario is also possible. When several computers search the solution space simultaneously, then a good upper bound UB may be found earlier than in the sequential run. As a result, some parts of the search tree which would have been visited in a sequential run, may be pruned in a parallel run. Such phenomena are known as performance anomalies in PBB [11,12]. In distributed systems other difficulties arise. For example, termination of the computation may be erroneously declared when some BBTtree node is lost. This may happen due to errors in communication, or as a result of transition state when some tree nodes are in the communication network, while computers have nothing to process.

Thus, the following issues must be taken into account in PBB:

1) *load balancing* is necessary to avoid idling or overloading the processors,
2) upper bounds must be globally communicated to prune unneeded nodes,
3) deploying and quick initiation of computation to avoid idling of the computers,
4) reliable termination of computation.

4.2 Branching Scheme

In the following subsections we outline our PBB implementation. The branching scheme can be seen as construction of all possible job permutations. Let AS be the set of already sequenced jobs. The jobs in set $TS = \mathcal{J} - AS$ remain to be sequenced. Suppose we have two sequences π, σ partitioning AS, i.e., job sets in π, σ are disjunctive and their sum is equal to set AS. Initially $\pi = \sigma = ()$, $AS = \emptyset$, and $TS = \mathcal{J}$. Pair (π, σ) is a BBTnode representing all the schedules starting with sequence π and finishing with sequence σ. Let $|x|$ denote the number of jobs in sequence x. *Height* of a BBTnode is the height of the subtree it represents, i.e. height of (π, σ) is $n - |\pi| - |\sigma|$.

Branching BBTnode (π, σ) consists in inserting unassigned jobs $J_j \in TS$ between sequences π and σ. Jobs J_j may be attached either to the end of π or to the beginning of σ, but only one option can be used to avoid generating the same sequences many times. Both assignments are verified for each job in TS, but only one assignment type is used in all the successors of (π, σ). The assignment which results in a greater number of the offspring nodes with their lower bounds smaller than the current upper bound UB is selected. If both choices are equivalent, then the assignment giving the smallest lower bound for any new node is chosen. Otherwise, all the jobs are arbitrarily attached at the beginning of σ.

The branching process is finished when a *complete* solution is achieved, i.e. when $TS = \emptyset$ or equivalently $|\pi| + |\sigma| = n$. Observe that the search method is exhaustive, i.e. all possible sequences may be enumerated (unless they are pruned), and no sequence is generated twice.

The search tree was explored in the Depth-First Least Lower Bound (DF/LLB) order, i.e. the newly branched BBTnodes (π', σ') were sorted according to non-decreasing values of lower bounds $LB(\pi', \sigma')$ and analyzed in this order before the older BBTnodes.

4.3 Bounding Techniques

Let us remind that BBTnodes with lower bound greater than or equal to the upper bound are not expanded. The upper bound UB is the schedule length of the best known solution of the problem. The initial UB value is the length of the schedule built by NEH heuristic [13]. NEH first sorts the jobs in the order of nonincreasing total processing times $\sum_{i=1}^{m} p_{ji}$. Then the first two jobs are scheduled for the minimum schedule length. Thus, a sequence of $l = 2$ jobs is constructed. For the given sequence of l jobs schedules with the $(l+1)$th job inserted between all the jobs in the sequence of length l, including the starting and the ending positions, are verified. The best schedule for $l + 1$ jobs is chosen and l is increased. This procedure is repeated until inserting all jobs, i.e. until $l = n$. The value of UB is updated and the new solution is recorded when a better solution is found in a leaf of the search tree.

Now we proceed to the methods of lower bound calculation. A BBTnode consists of two sequences (π, σ). The unscheduled tasks from set TS shall be either

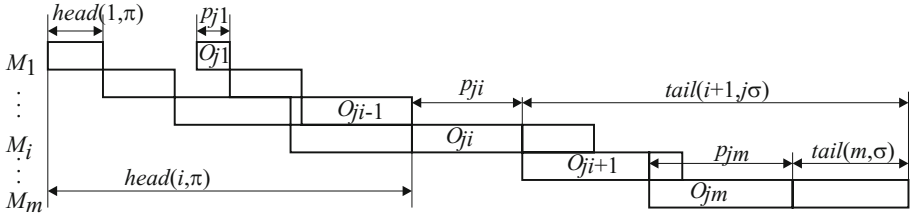


Fig. 2. Calculation of $head(i, \pi j)$ and $tail(i, j\sigma)$

appended to π, or attached at the beginning of σ. Suppose job J_j is inserted at the beginning of sequence σ, and the offspring node is $(\pi, j\sigma)$. Consider operation O_{ji} which is immediately followed by the operations in σ on machines $M_i, \ldots, M_m$ (cf. Fig.2). Operation O_{ji} and its successors in $j\sigma$ will be executed in time at least $tail(i, j\sigma)$. Let $tail(m+1, j\sigma) = tail(m, \sigma)$. The value of $tail(i, j\sigma)$ is calculated using the following formula:

$$tail(i, j\sigma) = \max\{p_{ji} + tail(i+1, j\sigma), tail(i, \sigma)\} \text{ for } i = m, \ldots, 1 \qquad (1)$$

Note that $tail(i, j\sigma)$ is tabulated after $tail(i+1, j\sigma)$. Analogously, assume $J_j \in TS$ is to be appended to π, and $(\pi j, \sigma)$ is the offspring. Operation O_{ji} is preceded by the operations in π on machines $M_1, \ldots, M_i$. Let $head(i, \pi j)$ denote the minimum time it takes to execute these operations. Let $head(0, \pi j) = head(1, \pi)$. The value of $head(i, \pi j)$ can be tabulated using the following formula:

$$head(i, \pi j) = \max\{p_{ji} + head(i-1, \pi j), head(i, \pi)\} \text{ for } i = 1, \ldots, m \qquad (2)$$

Again $head(i, \pi j)$ is calculated after $head(i-1, \pi j)$. Let π', σ' be the offspring sequences. If J_j is appended to π, then $\pi' = \pi j, \sigma' = \sigma$, otherwise J_j immediately precedes σ and $\pi' = \pi, \sigma' = j\sigma$. The first lower bound was calculated in $O(m)$ time from the formula

$$LB_1(\pi', \sigma') = \max_{i=1}^{m}\{head(i, \pi') + \sum_{j \in TS - \{J_j\}} p_{ij} + tail(i, \sigma')\} \qquad (3)$$

Also a second lower bound was initially used. It was based on the above defined $head(k, \pi), tail(l, \sigma)$, and the length of the schedule for tasks in set TS executed on any pair of machines M_k, M_l only. Though this lower bound effectively reduced search tree, it had higher complexity $O(m^2)$. Consequently, despite optimizations speeding computation of the second lower bound, it did not reduce overall running time.

4.4 Parallelizing for the Grid

In this section we describe control mechanisms and load balancing. Active nodes are stored in queues at each of the computers. An active BBTnode is a unit of load balancing.

Control Architecture. The control structure is a three-tier tree comprising a master server, slave servers and clients. There is one master server in the tree root. To avoid a communication bottleneck on the master server monitoring the whole computation, an intermediate layer of slave servers was introduced. A slave server manages a set of its clients. Except for the computation initiation the servers perform communications and load balancing. Clients compute and shift the load. All communications are performed on the paths up and down the tree.

The number of clients is equal to the number of CPU cores. Master server deploys a slave server for each 8 clients, and at least one slave server in each geographic location. The number of 8 clients per slave server was established experimentally in the preliminary runs on the SMP machine. The slave servers deploy their clients.

Both the master server, and the slave servers verify if the subordinate machines are *alive*. This is done by ping-like function during the BBTnode harvesting procedure (see the next paragraph). Let us observe that the timeout for a ping was 15s which is quite long period. A slave server or a client can be eliminated from the computing pool also if it causes a communication error, e.g., when a controlling computer tries to send to its subordinate some BBTnodes. If a computer does not respond or causes a communication error, then it is declared *dead*, removed from the computer pool and no longer used. The BBTnodes assigned to such a computer are moved back by the supervising server to the queue of active nodes to be reassigned. Computations finish when there are no BBTnodes to branch (see the next paragraph for details).

Load Balancing. At the start of the computation the master server branches the BB tree to the depth of two levels, creating $n(n-1)$ active nodes of height $n-2$, and sends one node to each slave server. Each slave server expands the received BBTnode by two additional levels, creating $(n-2)(n-3)$ new BBTnodes of height $n-4$, and sends one node to each of its clients. The servers record information on the forwarded nodes and their destinations. A client expands the received node to a full depth of the BB tree. If a client achieves a complete solution b which is better than the current upper bound, i.e. $C_{max}(b) < UB$, then b, UB are sent to the slave server. If the received upper bound UB is better than the old one, then the slave server sends it up to the master server, and down to its other client computers. Analogously, the master server sends a new better UB to the other slave servers, which forward it to their clients. A client or a slave server ignore the received upper bounds if these bounds are worse than the bounds known to the client or the slave server. A BBTnode a which has $LB(a) \geq UB$ (because a was enqueued when UB was bigger) is discarded when pulled from the queue for branching or load balancing.

If a client exhausts all its BBTnodes, then it requests one node from its slave server. The slave server records that the node previously sent to the client is fathomed, and assigns to the client a new BBTnode from its queue. If no active nodes are available at the slave server, then the slave server requests a node from the master server. The master server acts similarly if it has a BBTnode

available. If it has no BBTnodes available then the initial BBTnodes of height $n-2$ are distributed, and the computation is in the final stage. Then, the master server starts a harvesting procedure.

In the first phase of harvesting the master server requests half of the nodes from the slave servers. A slave server returns at most half of its BBTnodes but only if they have sufficient height. If the request is successful, then the received BBTnodes are sent by one to the waiting slave servers. If the request is not successful, then the second phase of harvesting is initiated. Namely, the master server sends to the slave servers requests for half of BBTnodes from the clients. If a client has BBTnodes of sufficient height then it sends them to its slave server. The slave server keeps half of the received nodes, and the other half is transferred to the master server, provided that they have sufficient height.

The height of the BBTnodes returned to the servers in the harvesting procedure is important for the PBB performance. On one hand, low BBTnodes allow for fine granularity load balancing because they represent smaller subtrees. On the other hand, low BBTnodes represent smaller computational demands. They can be fathomed faster, and clients return for new BBTnodes earlier than for high BBTnode. Hence, use of high BBT nodes reduces the number of communications. In the tests on an SMP machine the minimum heights of the returned nodes were set to 10 for slave servers, and 8 for clients. In the grid, the number of processors was bigger and the load transfers were too frequent overloading the computers with handling communications. Therefore, BBTnodes lower than 12 at the slave servers or 10 at the clients, were not returned in the grid runs.

If the master server request for BBTnodes at the second phase of harvesting fails, then it means that only low BBTnodes are held by the clients and slave servers (if any). In such a case a request to confirm the completion of the computations is issued by the master server to all live slave servers. The slave servers wait for a confirmation of the completion of the computation at the live clients. When a client exploits all its BBTnodes, then the confirmation is sent. If all its clients confirmed then also the slave server confirms completion to the master server. Finally, computations stop if all live slave servers confirm completion of the computation to the master server.

We observed that in the final stage of the computation the number of exchanged messages was intensively increasing. This was a result of exchanging control messages to achieve load balance with a small number of final BBTnodes. As they had low subtrees, clients quickly fathomed them and returned requests for new BBTnodes. Thus, the load balancing method needs better tuning for the final stage of computation.

5 Experiments

Running a PBB was not always as smooth as one could expect. Application deployment was time consuming and not always successful. To speed up the deployment process our PBB used parallel deployment of the clients by the slave servers. Not always have the machines declared available successfully deployed

Table 1. Example of the infrastructure deployed for instances Taillard 21,28,29

Location	No. of Computers	CPUs per computer
Amsterdam	20	2 x 1GHz
Supelec	33	2 x 3GHz
Lifl	53	2 x 2.4GHz
Inria Sophia	16	2 x 2GHz
Inria Sophia	16	2 x 933MHz
Inria Nef	32	2 x 2GHz

Table 2. Runtimes for Taillard 20×20 instances

Taillard instance	Runtime (grid) [s]	CPUs (grid)	Runtime (SMP) [s]
No. 21	435	370	1285
No. 22	219	361	499
No. 23	1746	352	16627
No. 24	234	361	502
No. 25	351	361	925
No. 26	515	361	1486
No. 27	607	352	2119
No. 28	148	370	102
No. 29	187	370	234
No. 30	139	361	108

the code. Consequently, our application quickly discarded slow and unreliable computers. For example, out of 2300 computers declared available, instances Taillard 21, 28, 29 were solved on grid shown in Table 1. Hence, our attempt in heterogeneous computing immediately reduced the heterogeneity of the platform for the benefit of speed and reliability.

It was possible to solve all the submitted 20×20 FS instances to optimality within the 1 hour limit imposed by the competition rules. The running times and numbers of CPUs are shown in Table 2. The last column of Table 2 give running times on SMP SunFire machine. The computing infrastructure comprised moderate number of processors from close locations. This allowed for using fewer but more reliable CPUs. The optimum solutions in Table 2, were obtained despite small number of CPUs, because the FS part in PBB was adequately implemented. It can be observed [4] that other teams of the Flowshop Challenge used a strategy oriented toward employing big number of CPUs. This strategy, however, turned out futile because using more CPUs did not result in solving the instances.

6 Conclusions

We presented PBB for permutation flowshop as a benchmark of maturity of grid computing and the supporting middleware in executing applications with unpredictable resource demands. Both the platform, and the middleware excelled well. Furthermore, qualitative conclusions can be drawn. Three elements constitute the parallel application described above: flowshop algorithm, PBB, and grid interaction algorithms. All the three elements had to be adequately addressed. It is not possible, for example, to ignore flowshop domain issues and rely solely on the sheer parallel processing power to solve the problem because the number of possible solutions is anyway too big. Yet, too complex domain-specific solutions turned out counterproductive. PBB must actively shift the work between the machines to account for irregular and unpredictable load distributions.

The grid part must monitor resource availability to avoid stalls in the computation. Though the techniques we presented are not new, only combined together were they able to produce a successful parallel application for hard problem on a difficult computing platform.

Acknowledgments. Research partially supported by Polish National Science Center grant N519 643340.

References

1. Bąk, S., Błażewicz, J., Pawlak, G., Płaza, M., Burke, E., Kendall, G.: A parallel branch-and-bound approach to the rectangular guillotine strip cutting problem. INFORMS J. on Computing 23, 15–25 (2011)
2. Clausen, J.: Branch and bound algorithms - principles and examples, Technical Report, Department of Computer Science, University of Copenhagen (1999)
3. Crainic, T., Le Cun, B., Roucairol, C.: Parallel Branch-and-Bound Algorithms. In: Talbi, E.-G. (ed.) Parallel Combinatorial Optimization, pp. 1–28. John Wiley & Sons (2006)
4. ETSI: 2nd Grid Plugtests Report (2006), http://www.etsi.org/website/document/plugtestshistory/2005/2ndgridplugtestsreport.pdf
5. Garey, M., Johnson, D., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1, 117–129 (1976)
6. Hejazi, S., Saghafian, S.: Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 43, 2895–2929 (2005)
7. Horn, J.: Bibliography on parallel branch-and-bound algorithms (1992), http://liinwww.ira.uka.de/bibliography/Parallel/par.branch.and.bound.html
8. Johnson, S.M.: Optimal two-and-three-stage production schedules with set-up times included. Naval Research Logistics Quarterly 1, 61–68 (1954)
9. Iyer, S., Saxena, B.: Improved genetic algorithm for the permutation flowshop scheduling problem. Computers & Operations Research 31, 593–606 (2004)
10. Kohler, W., Steiglitz, K.: Enumerative and iterative computational approaches. In: Coffman Jr., E.G. (ed.) Computer and Job-Shop Scheduling Theory, pp. 229–287. Wiley, New York (1976)
11. Lai, T.-H., Sahni, S.: Anomalies in parallel branch-and-bound algorithms. Communications of the ACM 27, 594–602 (1984)
12. Lai, T.-H., Sprague, A.: Performance of Parallel Branch-and-Bound Algorithms. IEEE Transactions on Computers 34, 962–964 (1985)
13. Nawaz, M., Enscore, E., Ham, I.: A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. Omega 11, 91–95 (1983)
14. Reeves, C., Yamada, T.: Genetic algorithms, path relinking, and flowshop sequencing problem. Evolutionary Computation 6, 45–60 (1998)
15. ProActive - Professional Open Source Middleware for Parallel, Distributed, Multicore Programming, http://proactive.inria.fr/
16. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
17. Taillard, E.: Scheduling instances (2008), http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

An Experimental Comparison of Load Balancing Strategies in a Web Computing Environment*

Joachim Gehweiler, Peter Kling, and Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute and Computer Science Department,
University of Paderborn, 33102 Paderborn, Germany
{joge, peter.kling, fmadh}@uni-paderborn.de

Abstract. Web Computing is a variant of parallel computing where the idle times of PCs donated by worldwide distributed users are employed to execute parallel programs. The PUB-Web library developed by us supports this kind of usage of computing resources. A major problem for the efficient execution of such parallel programs is load balancing. In the Web Computing context, this problem becomes more difficult because of the dynamic behavior of the underlying "parallel computer": the set of available processors (donated PCs) as well as their availability (idle times) change over time in an unpredictable fashion.

In this paper, we experimentally evaluate and compare load balancing algorithms in this scenario, namely a variant of the well-established Work Stealing algorithm and strategies based on a heterogeneous version of distributed hash-tables (DHHTs) introduced recently. In order to run a meaningful experimental evaluation, we employ, in addition to our Web Computing library PUB-Web, realistic data sets for the job input streams and for the dynamics of the availability of the resources.

Our experimental evaluations suggest that Work Stealing is the better strategy if the number of processes ready to run matches the number of available processors. But a suitable variant of DHHTs outperforms Work Stealing if there are significantly more processes ready to run than available processors.

1 Introduction

In recent years, Volunteer Computing and Grid Computing have received considerable attention. Web Computing is a variant of parallel computing that combines aspects of both volunteer and grid computing: Like Volunteer Computing, Web Computing means to utilize only the idle times on lots of donated PCs connected via the Internet to build up a virtual supercomputer (rather than connecting a few supercomputers or computing clusters). And like Grid Computing, Web Computing means to run coupled, massively parallel algorithms (rather than distributed data processing). The *Bayanihan* BSP implementation [9] is a first attempt to support bulk-synchronous parallel (BSP) programs in

* This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 31–40, 2012.

a Volunteer Computing context. A major problem for the efficient execution of such parallel programs is load balancing. The processes of the parallel programs have to be distributed among the processors such that all processes receive approximately the same amount of computing power. In the Volunteer Computing context, this problem becomes more difficult because of the dynamic behavior of the underlying "parallel computer": the set of available processors (donated PCs) as well as their availability (idle times) change over time in an unpredictable fashion. The *Bayanihan* BSP implementation uses a central master node that schedules the processes; all communication and checkpointing traffic passes through the master node.

Our PUB-Web library is a Web Computing library supporting the BSP programming style. In addition to the above mentioned library, PUB-Web allows very flexible load balancing, because it provides the possibility to migrate threads. Thus, processes can be stopped and finished elsewhere. In this paper, we evaluate and compare several load balancing algorithms in this scenario. They work in a distributed fashion and only need little non-local information. On the one hand, we examine a variant of the well-established Work Stealing algorithm. On the other hand, we consider three strategies based on a heterogeneous version of distributed hash-tables (DHHTs) introduced recently [5]. In order to run a meaningful experimental evaluation, we need, in addition to our Web Computing library PUB-Web, realistic models for the BSP programs to run and for the dynamics of the availability of the resources (idle times of the donated PCs). For the former, we use synthetic streams of jobs modeled according to insights from traces of big parallel machines. For the latter, we use real data collected during a profiling of hundreds of PCs at several places.

Our experimental evaluations suggest that Work Stealing is the best strategy if the number of processes ready to run matches the number of available processors. However, a suitable variant DHHTs outperforms Work Stealing if there are significantly more processes ready to run than available processors.

The remainder of this paper is organized as follows: in the next section, we give an overview of our PUB-Web library. In Section 3 we present our DHHT-based load balancers and describe the Work Stealing variant which we compare against. In Section 4 we present the experimental setup, which we use for our evaluation of the load balancing in Section 5. Section 6 concludes this paper.

2 PUB-Web

Our *Paderborn University BSP-based Web Computing (PUB-Web)* library [7,3] is realized as a peer-to-peer (P2P) system, consisting of a dynamically changing set of maybe worldwide distributed computers temporarily donated to be used for Web Computing. In addition, a few supernodes are employed for the management of the system. PUB-Web only utilizes the left-over computing power on the donated computers, in order not to disturb other activities on these machines. The donated computers may be very different (e.g., desktop PCs, notebooks, etc.), both w.r.t. their computing power and the dynamics of their availability

(i.e., their left-over computing power). Note that the availability depends on the way the computers are used by other activities of their users (e.g., regular or irregular patterns). This might be very different for different computers. For the remainder of this paper we refer to these other activities as *external workload*.

All peers of the PUB-Web network are allowed to run *processes* of *parallel jobs*. As a basic precondition for rebalancing the process-to-peer assignment on sudden changes in the external workload, PUB-Web supports thread migration. In order to continue the execution of a parallel job if one of the participating peers crashes, PUB-Web creates checkpoints at regular times and stores them at different nodes across the network.

PUB-Web supports the well-established *Bulk-Synchronous Parallel (BSP)* programming style [12,1]. This has been introduced by Leslie G. Valiant in order to simplify the development of parallel algorithms. It forms a bridge between the hardware to use and the software to develop by giving the developer an abstract view of the technical structure and the communication features of the hardware to use (e.g., a supercomputer with shared memory, a cluster of workstations, or PCs connected via the Internet). Jobs running in the PUB-Web system are *BSP programs*, which consist of a set of processes and a sequence of *supersteps* – time intervals bounded by a barrier synchronization. Within a superstep each process performs local computations and sends messages to other processes. When all processes have invoked the sync method and all messages are delivered, the next superstep begins and the messages sent during the previous superstep can then be accessed by its recipients. When run on PUB-Web, each superstep of a BSP program constitutes a load balancing problem for independent processes; the ability of PUB-Web to perform thread migration enables *preemption* of processes, so that they can be completed on another processor.

3 Load Balancing

In this section we give a description of the scheduling problem, discuss a Work Stealing variant, and three variants of our DHHT-based load balancer.

Problem Description. Our scheduling problem can be formulated as follows: We are given a *stream describing the availability* of the peers and a *stream of jobs* consisting of independent processes. For a time t, the *availability* of a peer is described by a value in $[0; 1]$ indicating the fraction of its computing power currently available. This value reflects removal / addition of peers (value changes to 0 / from 0 to something greater than 0) or the power left over beside the current external workload. This stream may show an unpredictably changing behavior. In order to reflect the heterogeneity of the hardware, we associate a static *weight* factor from $(0; \infty)$ describing the processor speed. This value is derived by a benchmark being part of PUB-Web. Thus, at any time, the *currently available computing power* of a peer can be expressed as the product of the peer's weight and its availability in a globally uniform way.

The supersteps of a BSP program are by definition subsequent, disjoint time intervals. Thus, scheduling a BSP program with ℓ supersteps is equivalent to

scheduling ℓ successive BSP programs with one superstep each, where the release time of the second program is identical to the finishing time of the first program, and so on. As a consequence, each job in our stream of jobs is described by a release time, the number of its processes, and their lengths. These lengths are assumed to be identical within one job. The processes to be executed at a given time may belong to different jobs and may therefore have different lengths. We consider all processes of all jobs to be equally important, independently of their length and job-parallelism.

Our optimization goal is to ensure that (i) all processes receive approximately the same amount of computing power, and (ii) the total processor utilization is maximized without violating condition (i). To this end, the PUB-Web system has to regularly (re-) assign all currently running processes to peers using its migration capabilities.

We now describe the load balancing algorithms evaluated and compared in this paper.

Work Stealing. The Work Stealing idea has been well studied over the past decades. In [8] for example, a Work Stealing strategy is presented for balancing the load of independent processes on a parallel computer, i.e., a set of homogeneous, fully available processors. In [2], a Work Stealing algorithm for well-structured, multithreaded computations is presented. We have examined the following variant: Consider a randomized, distributed setting, where each computing node maintains a FIFO queue. Whenever a processor becomes idle, it removes the first process from its queue and executes it. In case its queue is empty, the processor "steals" the first process from the queue of another, randomly chosen processor. When a job with parallelism k is released, its processes are added to the queues of up to k distinct, randomly chosen processors, with probabilities inverse-proportional to the current queue-sizes. This, and the fact that a processor only steals a process when its queue has run empty, helps to prevent unnecessary migrations. Using a queue instead of a stack and stealing processes from the head instead of the tail of the queues is useful for our BSP-scenario, because it makes sure that no jobs are disadvantaged. This algorithm can be implemented in an efficient and fully distributed fashion.

Distributed Heterogeneous Hash-Tables (DHHT (Rnd.)). In [5] we have proposed a novel distributed load balancer, which provides scheduling, migration, and fault tolerance for processes using *Distributed Heterogeneous Hash-Tables (DHHTs).* DHHTs are a heterogeneous generalization of the consistent hashing introduced by Karger et. al. [6]. In order to map all running processes to the active peers, first all peers are hashed uniformly and independently at random into the $[0;1)$-interval, which is interpreted as a unit ring (cf. Fig. 1). There is a linear function associated with each node, whose gradient is the inverse of the currently available (globally normalized) computing power. The lower envelope for the $[0;1)$-interval is defined, at any point x, as the peer whose linear function has the minimum value at point x; this way we assign sub-ranges of the $[0;1)$-interval to the peers. Finally, all processes are also hashed uniformly and independently at random into

the $[0;1)$-interval and are assigned to the peers associated with them through the lower envelope.

With this approach, the assignment of a process to a peer does not depend on the presence or assignment of other processes. Thus, using pseudo-random or deterministic hash functions, only the weight and availability information about the active peers need to be communicated.

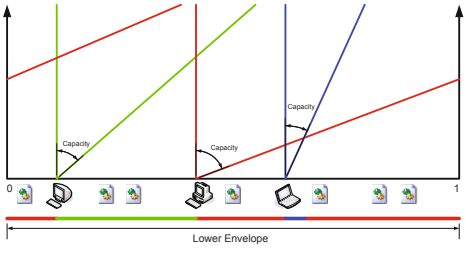


Fig. 1. Load balancing using DHHT

Concerning the balancing quality, it is shown in [10] that a process is assigned to peer i with probability $p_i \leq \frac{w_i}{W-w_i}$, where w_i is the available computing power of peer i and $W = \sum_i w_i$.

Our implementation uses SHA-1 as a pseudo-random uniform hash function. We refer to this DHHT variant as "DHHT (Rnd.)" in the remainder of this paper.

Double Hashing (DHHT (Dbl.)). In [10] several possible improvements are discussed; among them a double hashing technique, for which it is shown that a process is assigned to peer i with probability $(1-\sqrt{\epsilon}) \cdot \frac{w_i}{W} \leq p_i \leq (1+\epsilon) \cdot \frac{w_i}{W}$. We also implemented this variant and refer to it as "DHHT (Dbl.)" in the following.

Deterministic Hashing (DHHT (Mult.)). It is well-known that when hashing peers uniformly and independently at random, the longest interval has length $\Theta(\frac{\log n}{n})$ with high probability, and the shortest one has length $\Theta(\frac{1}{n^2})$ with constant probability. Thus, the ratio of the longest interval to the shortest one, called *smoothness*, is $\Theta(n \cdot \log n)$ with high probability.

In order to overcome this drawback, we implemented another, deterministic DHHT variant based on multiplicative hashing: In [11] it is shown that when consecutively adding the points $i \cdot \Phi \mod 1$, $i = 0, 1, 2, \ldots, n$, to $[0,1)$, where $x \mod 1 := x - \lfloor x \rfloor$ and $\Phi := \frac{1}{2} \cdot (\sqrt{5}-1)$, a smoothness of Φ^2 is obtained for any $n \in \mathbb{N}$. In the following, we refer to this variant as "DHHT (Mult.)".

4 Experimental Setup

In order to perform a meaningful experimental evaluation of the load balancing, we need realistic information about the capabilities of the simulated hardware, about the dynamics of the external workload on the peers, and about the parallel programs to run.

4.1 Capabilities of the Processors

We use the following weight factors: Approximately 15% of the factors are in each of the intervals $[0.6;1.5)$, $[1.5;2.5)$, $[2.5;4)$, and $[4;5)$; the remaining 40%

are in the range $[5; 6.2]$. These values are derived from a collection of 100 computers donated to PUB-Web during the European project "AEOLUS" (FP6 IST-15964).

4.2 Dynamics of the External Workload

In order to collect real usage data for the evaluation of our load balancers, we did a detailed, long-term analysis of the availability of several hundred computers.

In total we have analyzed the CPU usage on more than 250 PCs for several months. Among these PCs there are different sets of office computers, PCs in computer pools available to students, notebooks of employees, and both home PCs and notebooks of individuals. The examined computers are very heterogeneous with respect to their hardware and operating system.

In the following, we analyze the collected data with respect to these criteria:

Availability. How often a computer is switched on, and how much idle time is available, affects the total amount of computing power available in the PUB-Web network. This parameter describes how much computing power is available on a computer on average.

Diversity. If all computers in the PUB-Web network would be switched on and off at the same time, and would have the same loss or gain of computing power at the same times, load balancing would be trivial because all running processes would experience the same slowdown or speedup. However, that is usually not the fact. The diversity parameter describes how different the availability curves of the computers behave.

Uniformity. If the availability of a computer is rather stable (at any level) for long time intervals, the load balancer rarely has to rebalance the assigned processes. This parameter describes how often the availability of a computer significantly changes (high uniformity = rare changes).

Cumulative Pattern. When averaging over a big amount of computers, the aggregated data may show a certain pattern even if the diversity is high and the uniformity is low. Typical patterns are daytime / nighttime or workday / weekend differences, and may help to improve the load balancing.

Analyzing the collected data, we identify distinguishing parameters for three types of computers: office PCs, notebooks, and pool PCs used by students (cf. Table 1).

Table 1. Characterization of the availability

	Office PCs	Notebooks	Pool PCs
Availability	high	low	medium
Diversity	low	high	medium
Uniformity	high	low	low
Daytime Pattern	significant	significant	weak
Weekend Pattern	significant	significant	no

When the computers are switched on, the availability is very high: above 90% with a probability of approximately 90% for office PCs and pool PCs; and above 80% with a probability of approximately 75% for notebooks. If additionally taking into account the times, when the computers are switched off, we have an average availability of approximately 70% for office PCs, 45% for pool PCs, and 30% for notebooks.

Not surprisingly, the office PCs are used in a quite homogeneous way with respect to both time of usage and kind of usage, leading to a low diversity, a high uniformity, and clear daytime and workday patterns. Notebooks are often used in a mixed fashion for different purposes, leading to a high diversity and low uniformity; however, they still show clear daytime and workday patterns on average. The pool PCs are used by lots of different students for very heterogeneous tasks. Despite this fact, they show only a medium diversity; but as expected, they have a low uniformity and not even a clear daytime pattern.

For our evaluation of the load balancers, the diversity property will be most interesting as a high diversity generates a lot of imbalance.

4.3 Properties of the Job Stream Model

Beside the parameters for the hardware capabilities and the external workload, where we will use the collected data as input for our experiments, we will create synthetic job input streams because we do not have access to traces of comparable Web Computing systems. As already outlined in Section 3, a job stream consists of a sequence of jobs with certain release times, parallelism, and lengths.

The release times of the jobs are chosen uniformly and independently at random within the total simulation duration, except for a short warm-up phase at the beginning of the experiments. Realistic values for the length of a superstep in a coarse-grained BSP program are lower bounded by a couple of minutes (otherwise the synchronization overhead becomes unreasonably large) and upper bounded by roughly one hour (otherwise communication would occur unrealistically rarely). Remember from Section 3 that a BSP program consisting of ℓ supersteps is splitted into ℓ successive jobs. Thus, our jobs have lengths of 5, 10, 15, 20, 30, 45, or 60 minutes (on a CPU with weight factor 1 and 100% availability). For each job, one of these lengths is randomly chosen, where we use probability distributions derived from traces of supercomputers; in particular, we use the traces of three Linux clusters and two IBM systems available through the *Parallel Workloads Archive*[1] to obtain different realistic assumptions about the job lengths and parallelism. While we only use the probability distribution of the job lengths to derive realistic quantities for our model, we directly copy the probability distribution of the parallelism; all parallelism values occurring are powers of 2, up to 1024, where small values are much more likely than big ones.

The total number of jobs is chosen such that there are sufficiently many processes ready to run in all of our experiments. Since all of our experiments are performed under a fully loaded or slightly overloaded system, and because we need to control the load as the system would collapse at some time under very heavy overload, we keep all released processes in a ready queue if the total system load exceeds a certain limit. This limit is given by the number of peers in the PUB-Web network times a so-called *overload factor*. This procedure allows for a fair comparison of the load balancers because each load balancer has to

[1] `http://www.cs.huji.ac.il/labs/parallel/workload/index.html`

deal with the same total system load, independently of its throughput, and gets exactly the same sequence of jobs as input stream.

Though the job streams are generated in a randomized fashion according to the probability distributions of the parallelism and job lengths, we ensure, by using random seeds, that exactly the same job stream is used in a series of experiments for comparison of the particular load balancers.

5 Evaluation of the Load Balancing

Using the collected data and the job stream model described in the previous section, we are now able to conduct reproducible experiments: From each of the three profile groups (office computers, PCs in computer pools, notebooks) we use an interval of two weeks of collected data as input for our simulations. In total we simulate more than 10^8 processes in more than 700 experiments. The complete evaluation can be found in [4]. In order to separate the influences of the external work load from potential drawbacks of the load balancers, we perform three types of experiments:

A: To focus on the dynamics of the external workload, we feed the load balancers with the measured availability in the different profile groups, but do not generate additional load imbalances by a varying number of processes in the system; instead, we consider $x = 1, 2, 4$ times as many processes as processors, which are simultaneously released at the beginning of the experiment and run throughout the whole duration of the experiment. Thus, we measure with our experiments how well processes are distributed among the available peers, and how often migrations occur.

B: In order to focus on the processing of the jobs, we fix the availability of all peers to 100% throughout the whole experiment and feed the load balancers with the different synthetic job streams. Thus, we measure how fast and how fair the processes are executed.

C: We run the experiments of type B again, but now apply the same profiles of the external workload as in the experiments of type A. Thus, we measure the quality of our load balancers under dynamically changing availabilities of the processors and for realistic job streams.

Experiments of Type A. Since we run a fixed number of processes throughout the whole duration of the experiment, this setup is not suitable for Work Stealing. Thus, we only compare our three DHHT variants.

Due to space limitations we focus on the most interesting results that our evaluation reveals: DHHT (Mult.) yields 100% utilization of the available CPU time in all cases, whereas DHHT (Rnd.) and DHHT (Dlb.) only achieve around 70% for $x = 1$ and improve to 90%–99% (depending on the external workload) for $x = 4$. The balancing quality for DHHT (Dbl.) and DHHT (Mult.) is at a comparable level and significantly better than for DHHT (Rnd.). The number of migrations is at a comparable level for DHHT (Rnd.) and DHHT (Dbl.) and is significantly lower for DHHT (Mult.); even for the notebooks profile group,

where the most migrations occur on average, we have less than one migration per process and hour.

Experiments of Type B. In this set of experiments we perform two runs, one with overload factor 1 (i.e., with an ideal total system load) and one with factor 4.

In the following, we call the factor between the actual average runtime and the length of the processes the *process stretch factor*; analogously, we define the *job stretch factor*.

Under ideal system load conditions we are not able to keep up with the Work Stealing algorithm; DHHT (Mult.) is the best of our candidates, which only suffers a performance drawback of a factor of approximately 1.25–2 (depending on the job input stream) in terms of the job stretch factor.

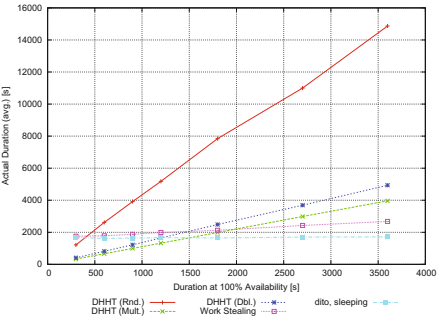


Fig. 2. Average runtime of processes under overload

But under the overloaded conditions, Work Stealing suffers a performance loss of a factor of 3–6 (depending on the job input stream) with respect to the job stretch factor. Exemplarily for one of our experiments, Fig. 2 shows that the Work Stealing curve still is the one that grows least, but is notably shifted upwards. At first glance, this may appear strange; but when looking at the Work Stealing sleeping curve, we notice that all processes wait in the queues approximately for the same time, independently of their lengths. As a consequence, 5-minute-jobs take more than 5 times as long as when executed using the DHHT (Mult.) load balancer, and 1-hour-jobs are less than 1.5 times faster, leading to the overall performance loss for Work Stealing in average.

Experiments of Type C. Finally, we repeat the experiments of type B again, but apply the three external workload profiles in order to obtain completely realistic circumstances. Due to space limitations we focus on the main results:

When the external workload has a low or medium diversity, the results are very similar to the ones obtained for our experiments with 100% availability; in particular, DHHT (Mult.) suffers a performance drawback of a factor of only approximately 1.25–2 against Work Stealing in the ideal system load case, and it outperforms Work Stealing under overloaded conditions by a factor of approximately 3–7. On a high diversity in the external workload, DHHT (Dlb.) performs better than DHHT (Mult.) and is able to keep up with Work Stealing up to a factor of approximately 2 in the ideal system load case; under the overloaded conditions, also DHHT (Dbl.) is best, outperforming Work Stealing by a factor of approximately 2.

6 Conclusion

Our evaluation reveals: as long as the total system load matches almost ideally the number of available processors, the Work Stealing algorithm performs best. However, on overloaded systems our algorithms outperform Work Stealing by a factor of 2–7 (depending on the diversity of the external workload).

Further future work could include the research of a sophisticated algorithm which switches between the Work Stealing and DHHT load balancers depending on the overall job load, the diversity of the external workload, or other criteria such as the desired fairness level (Work Stealing delays all jobs approximately equally, whereas DHHT delays the jobs proportionally to their length). Enhancements of the DHHT load balancer could include a multi-level load balancing in order to additionally consider further criteria beside the computing power, such as the network bandwidth for example.

References

1. Bisseling, R.H.: Parallel Scientific Computation: A Structured Approach using BSP and MPI. Oxford University Press (2004)
2. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), pp. 356–368 (1994)
3. Bonorden, O., Gehweiler, J., Meyer auf der Heide, F.: A Web Computing Environment for Parallel Algorithms in Java. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 801–808. Springer, Heidelberg (2006)
4. Gehweiler, J.: Peer-to-Peer Based Parallel Web Computing. Ph.D. thesis, Heinz Nixdorf Institute, Paderborn, Germany (2011)
5. Gehweiler, J., Schomaker, G.: Distributed load balancing in heterogeneous peer-to-peer networks for web computing libraries. In: Proceedings of the 10th Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 51–58 (2006)
6. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the 29th Symposium on Theory of Computing (STOC), pp. 654–663 (1997)
7. The Paderborn University BSP-based Web Computing (PUB-Web) library (October 2011), http://pubweb.cs.uni-paderborn.de/
8. Rudolph, L., Slivkin-Allalouf, M., Upfal, E.: A simple load balancing scheme for task allocation in parallel machines. In: Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 237–245 (1991)
9. Sarmenta, L.F.G.: An Adaptive, Fault-Tolerant Implementation of BSP for Java-Based Volunteer Computing Systems. In: Rolim, J.D.P. (ed.) IPPS-WS 1999 and SPDP-WS 1999. LNCS, vol. 1586, pp. 763–780. Springer, Heidelberg (1999)
10. Schindelhauer, C., Schomaker, G.: Weighted distributed hash tables. In: Proceedings of the 17th Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 218–227 (2005)
11. Swierczkowski, S.: On successive settings of an arc on the circumference of a circle. Fundamenta Mathematicae 46, 187–189 (1958)
12. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM 33(8), 103–111 (1990)

A Grid Scheduling Based on Generalized Extremal Optimization for Parallel Job Model

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

[1] Siedlce University of Natural Sciences and Humanities
Computer Science Department
3 Maja 54, 08-110 Siedlce, Poland
[2] Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw, Poland
[3] Cardinal Stefan Wyszynski University in Warsaw
Dep. of Mathematics and Natural Sciences
Woycickiego 13, 01-938 Warsaw, Poland

Abstract. In this paper we propose an efficient parallel job scheduling algorithm for a grid environment. The algorithm implies two stage scheduling. At the first stage, the algorithm allocates jobs to the suitable machines, where at the second stage jobs are independently scheduled on each machine. Allocation of jobs on the first stage of the algorithm is performed with use of a relatively new evolutionary algorithm called Generalized Extremal Optimization (GEO). Scheduling on the second stage is performed by some proposed heuristic. We compare GEO-based scheduling algorithm applied on the first stage with Genetic Algorithm (GA)-based scheduling algorithm. Experimental results show that the GEO, despite of its simplicity, outperforms the GA algorithm in all range of scheduling instances.

Keywords: grid computing, evolutionary algorithm, generalized extremal optimization, parallel job.

1 Introduction

Distributed computing has recently become popular technique to provide high performance computing for computationally intensive applications. This term is often described as "the Grids" or "Grid Computing". In grid computing there are multi-institutional virtual organizations with shared and coordinated resources. As resource we can define direct access to computers, software, data, or other resources, as it is required by a range of collaborative problem-solving [3].

Many studies propose the distributed management system [1] against the centralized scheduling [2]. There are also combination of distributed and centralized management [10]. It can be characterized by a hierarchical multilayer resource management [6]. The first (higher) layer is often assigned to a global grid scheduler. In this layer jobs are scheduled among the machines in the grid. At the second layer (lower) a local management system exists, which schedules assigned jobs on local machine.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 41–50, 2012.

The idea of grid computing has forced development of new algorithms of management of a large number of heterogeneous resources. The execution of the user's application must satisfy both job execution constraints and system policies. Scheduling algorithms applied to the traditional multiprocessor systems are often inadequate to grid systems. On the other hand, many algorithms [4], [5] have been adapted to the grid computing. However, there are many open problems in this field, including the consideration of multilayered system in the scheduling process.

This work is related to offline scheduling problem in grid computing and its resolution using meta-heuristic approaches. These algorithms are often used to solve a class of NP-hard problems. The scheduling problem belongs to this class. In this paper we will show the usefulness of meta-heuristic approaches for the design of efficient grid schedulers. We propose a relatively new meta-heuristic called GEO to solve the scheduling problem [8].

The paper is organized as follows. In the next section we define both the grid model and the scheduling problem. Section 3 presents the concept of GEO algorithm and its application for the scheduling problem. In Section 4 we present GA-based scheduling algorithm. Section 5 describes local scheduling algorithm. Next, in Section 6 we analyze experimental results comparing the use of GEO and GA-based scheduling algorithms. Last section contains conclusions.

2 Grid Model and Scheduling

2.1 Model

The grid model is defined as follows [9]. A grid system consists of a set of m parallel machines M_1, M_2, ..., M_m. Each machine M_i has m_i identical processors, called also the size of machine M_i. Fig. 1(a) shows an example of the grid system.

In the grid system there is a set of n jobs J_1, J_2, ..., J_n. A job J_j is described by a triple $(r_j, size_j, t_j)$. The release time r_j can be defined as the earliest time when the job can be processed. In this model we assume $r_j >= 0$. The size $size_j$ is referred to the processor requirements. It specifies a number of processors required to run the job J_j within assigned machine. We can define this as *degree of parallelism* or a *number of threads*. All threads of the job must be run at the same time and the same machine. The t_j is defined as the execution time of the job J_j. Fig. 1(b) shows an example of the job.

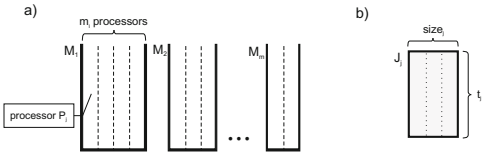


Fig. 1. Example of the grid system (a) and the job model (b)

Then, the $w_j = t_j * size_j$ denotes the *work* of job J_j. A machine executes a job of the $size_j$ when $size_j$ processors are allocated to it during time t_j. We assume that job J_j needs to be allocated only within assigned machine M_i. In other words, the job cannot be divided into small parts and executed on two or more machines simultaneously. Jobs must not be reallocated from different machines. The machine has to be capable to allocate the job to its processors, so $size_j <= m_i$ must meet.

Let us denote S as a schedule. As S_i we denote the schedule on machine M_i. The completion time of jobs on machine M_i in the schedule S_i is denoted by $C_i(S_i)$. We consider a minimization of the time $C_i(S_i)$ on each machine M_i over the system in such a way that the makespan is defined as

$$C_{max} = max_i(C_i(S_i)). \tag{1}$$

The purpose of the scheduling is to distribute jobs among the machines and schedule them to minimize a makespan C_{max}.

2.2 Two-Stage Scheduling

The proposed scheduling is the two-stage algorithm. An important role of the scheduling process is an appropriate allocation of jobs on machines. We consider a system with a different number of processors in machines. It forces to use an algorithm which globally distributes jobs among the system. Each job should be allocated in such a way that the makespan C_{max} is minimized.

According to the model we consider the first and the second stages of the scheduling. At the first stage the scheduling algorithm allocates globally jobs to suitable machines. At each step of the algorithm jobs are reallocated to machines with a lower utilization. The algorithm compares time of the schedule with a previous one. Then choose and reallocate the job to another machine, where the time can be minimized. At the second stage we apply a local scheduling algorithm. It schedules jobs in a particular machine.

3 Global Scheduling with Generalized Extremal Optimization Algorithm

3.1 The Bak-Sneppen Model and Its Representation in Job Allocation

At the first stage a meta-heuristic algorithm is applied. We propose a relatively new evolutionary algorithm called GEO. The idea of this algorithm is based on the Bak-Sneppen model [8]. 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 shows 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 affect the entire system. The probability distribution of these avalanches is described by a power law in the form:

$$p_i = k_i^{-\tau}, \tag{2}$$

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 when $\tau \to \infty$, 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(a) shows the population of species in the Bak-Sneppen model [8] and the Fig. 2(b) presents the idea of GEO algorithm of the grid scheduling.

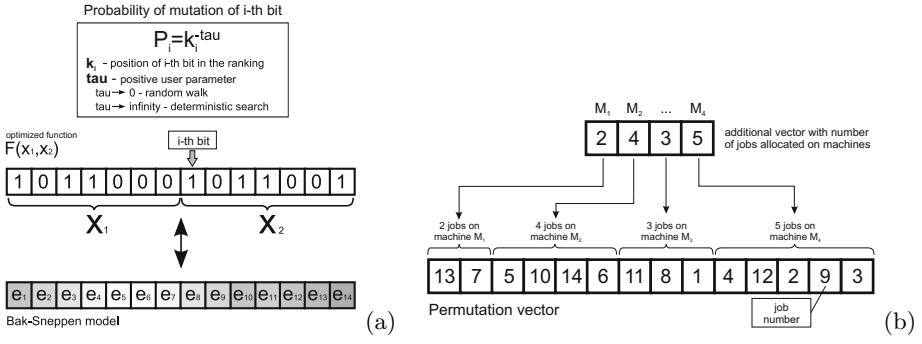


Fig. 2. Population of species in the Bak-Sneppen model and its correspondence in the GEO algorithm (a). Representation of grid model in Bak-Sneppen model (b).

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 scheduling problem presented in this paper contains one string of n numbers. The length of the string is equal to the total number of jobs in the grid system. Fig. 2(b) (upper part) shows an example of the GEO string which presents proposed allocation of jobs to machines in the grid. The job numbers are placed in a permutation form. This form defines the order of jobs executing by a local scheduling algorithm. To assign jobs to machines we set the additional vector (top vector on the Fig. 2(b)) showing the number of jobs allocated to corresponding machines. One can see that, for example, two jobs were allocated to machine M_1 and these are jobs 13 and 7. 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-Based Scheduling Algorithm

The GEO was originally presented by Sousa and Ramos [8]. In the algorithm each bit (species) is forced to mutate with a probability proportional to the fitness. It is a value associated with a given combination of bits of the GEO string, related to a problem. 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 corresponding to a current solution of a problem. The fitness can gain or lose 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. Since this moment the probability of mutation p_i of each i-th bit placed in the ranking can be calculated by Eq. 2 described above. In our problem

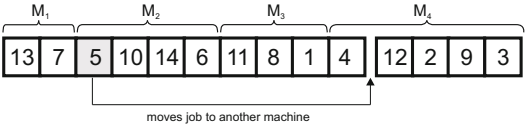


Fig. 3. Type of mutation used by GEO

we use integer numbers indicated jobs, instead of bits. Therefore, we propose another type of mutation operator adapted to our problem, called transposition. In Fig. 3 we present this type of mutation which is used by GEO. It assumes that a selected job migrates to another machine. At first, a job k (job 5 in Fig. 3) is randomly selected. The chosen job is moved to a machine having the shortest total time of execution of allocated jobs (machine M_4 in Fig. 3).

The GEO-based scheduling algorithm can be presented as the Algorithm 1.

Algorithm #1. GEO-based scheduling algorithm

1. Initialize randomly a permutation string of the length L that encodes n jobs.
2. For the current configuration C of jobs, calculate the value V corresponding to the objective function (1) and set $C_{best} = C$ and $V_{best} = V$.
3. For each job i do
   (a) mutate (transpose) each job and calculate the objective function value V_i of the string configuration C_i,
   (b) set the job fitness F_i as $(V_i - R)$, where R is a positive constant. It serves only as a reference number and can assume any value. The job fitness indicates the relative gain (or loss) that is a result of mutating the job.
   (c) return the string to its previous state.
4. Rank the N jobs according to their fitness values, from $k = 1$ for the least adapted job (on the top of the rank) to $k = N$ for the best adapted (with lowest ranking). In a minimization problem higher values of F_i are on the top of the rank. If two or more jobs have the same fitness, rank them in random order, but follow the general ranking rule.

5. Choose with an equal probability a job 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 job is mutated.
6. Set $C = C_i$ and $V = V_i$.
7. If $F_i < F_{best}$ then set $F_{best} = F_i$ and $C_{best} = C_i$.
8. Repeat steps 3 to 7 until a given stopping criteria is reached.
9. Return C_{best} and F_{best}.

4 Global Scheduling with GA

GA is a search technique used to find an approximate solution in optimization and search problems. It is a particular class of evolutionary algorithms (EA) that uses mechanisms inspired by evolutionary biology. The algorithm operates on a population of chromosomes coding potential solutions. Chromosomes usually are strings of bits. The fitness function is computed for each individual (chromosome). In the scheduling problem the fitness function is calculated as a makespan C_{max}. The Algorithm #2 presents the scheduling algorithm.

Algorithm #2. GA-based scheduling algorithm

1. Create an initial population of individuals (solutions of the problem)
2. Evaluate the fitness of each individual in the population (calculate the makespan)
3. Repeat
   (a) Use the Roulette Wheel operator to select individuals for reproduction
   (b) Apply operators: crossover and mutation to generate new solutions
   (c) Evaluate fitness of new individuals
   (d) Replace the population with new individuals
   Until stop condition is satisfied

In our problem strings are permutations of jobs similar to GEO string. As contrasted with GEO, which operates on one string, this algorithm operates on a set of strings.

5 Local Scheduling Algorithm

5.1 List Scheduling Algorithm

List scheduling is a heuristic technique encountered in scheduling algorithms [7]. In the first part of a list algorithm jobs J are sorted according to a priority scheme. In the second part, each job of the list is successively scheduled on a chosen machine. Usually, the chosen machine is the one that allows the earliest start time of the job. Algorithm #3 outlines the simplest form of list scheduling:

Algorithm #3. List scheduling algorithm

1. Sort jobs J into list L, according to priority scheme.
2. for each $J_i \in L$ do
   (a) Choose a machine M for J_i.
   (b) Schedule J_i on M.
   end for

5.2 Local Scheduling Algorithm

The algorithm of local scheduling allocates jobs within a particular machine. The main idea of this algorithm is to arrange jobs in such a way to minimize the area of empty space of the schedule corresponding to non-busy period processors time. Jobs assigned to a given machine are ordered by the global scheduler (permutated substring). In a local scheduler we use a simple list algorithm. At the beginning jobs J are sorted according to their work w. If some jobs have the same size of the work then they are ordered by a sequence given by a global scheduler.

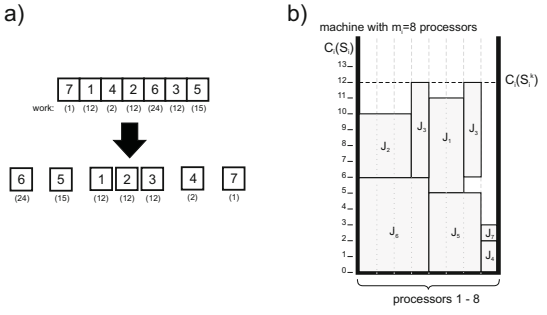


Fig. 4. A schema of local scheduling. The representation of the solution and its constructing (a) and allocation in the machine (b).

Fig. 4 shows an example of local scheduling. It is assumed that a subset (7,1,4,2,6,3,5) of seven jobs was assigned to a machine (see Fig. 4a, upper). Let us assume that each job consists of a number of threads: job J_7 consists of 1 thread, J_1 consists of 2 threads, J_4 consists of 1 thread, J_2 consists of 3 threads, J_6 consists of 4 threads, J_3 consists of 2 threads and J_5 consists of 3 threads. We calculate for each job its work. Let us assume that work for this subset is as follows: $w_7 = 1$, $w_1 = 12$, $w_4 = 2$, $w_2 = 12$, $w_6 = 24$, $w_3 = 12$, $w_5 = 15$. These jobs are next sorted in descending order according to their work w (see Fig. 4a, down). This sorted substring is used directly to assign them to processors of the machine. The job with the largest work is assigned first to processors. Next, the job J_5 is assigned to processors, and the remaining jobs are assigned in the following order: J_1, J_2, J_3, J_4, J_7. In such a way we obtain the schedule S_i^k with the makespan $C_i(S_i^k)$. As S_i^k we denoted actual schedule on the machine M_i.

6 Experimental Results

In this section we show the performance of GEO scheduling algorithm for the scheduling problem defined in this paper. At the first experiment we use some instances of scheduling problem to compare GEO with a list algorithm. In Tab. 1 we show the results obtained by these algorithms.

Table 1. Comparison of the makespan obtained by the GEO algorithm and list algorithm for some instances of the problem

Machine set Jobs set	*11_machines_1_2_4* *22_jobs*	*11_machines_1_2_4* *32_jobs_1_1*	*4_machines_4_8* *100_jobs_3_2*	*4_machines_4_8* *1000_jobs_3_2*
GEO	2	2	35	317
List algorithm	2	3	43	339

One can see that the GEO algorithm can find better solutions than list scheduling algorithm. For two first instances the optimal makespan is equal to 2. For the first instance (22 jobs with $size_j$ of threads in the range 1..4) both algorithms found an optimal result. When we used the second instance with the 32 jobs with $size_j$ threads equal to 1 only GEO found the optimal solution. We also test algorithms on randomly generated jobs sets. The sets of instances were generated with the following parameters: the number of jobs N_j, the average execution time of a job T_{avg}, and the average required number of processors S_{avg} (sets are noted as: N_j_jobs_T_{avg}_S_{avg}). It was assumed that the release time r was equal to 0 for both sets. In this case GEO found the better solution in all range of the scheduling instances.

In the second experiment we compare GEO with GA metaheuristic. Initially we show a typical run of GEO and GA algorithm (see Fig. 5). We run these algorithms for a 500 jobs set. One can see that during the first generations the GA algorithm quickly improves the solution, but later searching for new solutions goes slowly. The GEO, in opposite to the GA, starts with a relatively worse solution. However, it consistently improves solutions and quickly finds a solution outperforming ones presented by the GA. Calculation of the makespan is the main source of the time complexity of considered algorithms, and the number of evaluations (iterations of the algorithm) of the makespan in both algorithms may be different. We assumed an equal number of evaluations of the makespan for both algorithms.

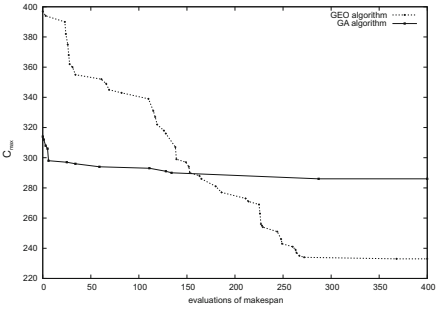


Fig. 5. Typical run of GEO and GA algorithm for 500 jobs set (T_{avg}=5, S_{avg} = 4) in 4 (with 8 and 16 processors) machines environment

In this experiment we use three sets of jobs (see Tab. 2) consisting of the number of jobs (N_j) ranging from 100 to 1000. The average execution time T_{avg} was set to 3, the average required number of processors S_{avg} was set to 2. For each set of jobs the problem of scheduling with use of 4 and 8 processors was considered. In the GEO we used the following parameters in the experiment: $\tau = 4.0$, a number of iterations of the algorithm $N_{itGEO} = 500$. For GA we set a population size $P_{GA} = 200$, mutation probability $p_m = 0.1$, crossover probability $p_c = 0.75$. The number of iterations was set in range $N_{itGA} \in \{250..2500\}$ (for an equal number of the evaluation of the makespan in both algorithms). We repeated each instance of the experiment 20 times. Tab. 2 shows the results

Table 2. Comparison of the makespan (in bold) obtained by GEO and GA for machines with use of 4 and 8 processors. The model of jobs assumes the average time T_{avg} equal to 3 and the average threads S_{avg} equal to 2. The average makespan is in italic text. In rounded brackets the standard deviation is given.

	4 machines		**8 machines**		**16 machines**		**32 machines**	
	4_machines_4_8		*8_machines_4_8*		*16_machines_4_8*		*32_machines_4_8*	
Jobs set	GEO	GA	GEO	GA	GEO	GA	GEO	GA
100 jobs	**35**	**36**	**20**	**22**	**11**	**13**	**8**	**9**
	35.1	*43.5*	*20.75*	*26.05*	*11.5*	*14.65*	*8.4*	*9.95*
	(0.3)	(5.7)	(0.4)	(2.1)	(0.6)	(1.3)	(0.5)	(0.8)
500 jobs	**147**	**181**	**81**	**96**	**39**	**49**	**22**	**30**
	147.2	*197.15*	*81.3*	*105.75*	*39.5*	*56*	*23*	*32.7*
	(0.4)	(9.2)	(0.4)	(5)	(0.6)	(2.7)	(1.4)	(2)
1000 jobs	**317**	**408**	**174**	**220**	**82**	**112**	**46**	**53**
	326.55	*440.7*	*179.75*	*232.4*	*92.85*	*119.9*	*54.8*	*66*
	(12.2)	(17)	(8.9)	(7.4)	(10)	(4.4)	(5)	(4.1)

of the experiment. We start from small instances of the problem. In the table we present the minimal time (makespan) obtained by algorithms, and in the brackets the standard deviation calculated for 20 runs of the algorithm. One can see that for 100 jobs the results are similar for both algorithms, but GEO slightly outperforms GA. The standard deviation for the GA is higher than for the GEO. This is visible especially for small machine sizes. For instances with use of 1000 jobs the standard deviation is smaller for GA than for GEO, however, the makespan is significantly smaller for GEO. This algorithm searches the solution more permanently and precisely. As we can notice, GEO can find appreciably better outcomes than GA. These results can be explained by behavior of GEO and GA algorithms. GEO finds the solution by calculating the makespan for each job from one string population and tend to accept strings for which mutation of job gives better result. In GA we calculate fitness function for a population of individuals. In GA mutation only maintains genetic diversity from one generation to another. Mutation in GEO is more meaningful. GEO finds a solution more consequently by precisely valuation of jobs and choosing the most suitable solution in the current step of the algorithm.

7 Conclusions

In this paper we have proposed a two-stage scheduling algorithm, where we used the relatively new meta-heuristic called GEO to solve the scheduling problem. We compared our results obtained with use of GA. We have shown that GEO-based scheduling algorithm outperforms GA-based scheduling algorithm in term of the makespan.

References

1. Ernemann, C., Yahyapour, R.: Applying Economic Scheduling Methods to Grid Environments. In: Grid Resource Management - State of the Art and Future Trends, pp. 491–506. Kluwer Academic Publishers (2003)
2. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R.: On Advantages of Grid Computing for Parallel Job Scheduling. In: Proc. of 2nd IEEE Int. Symposium on Cluster Computing and the Grid, pp. 39–46 (2002)
3. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal Supercomputer Applications. 15(3) (2001)
4. Ghafoor, A., Yang, J.: A Distributed Heterogeneous Supercomputing Management System. Computer 26(6), 78–86 (1993)
5. Hall, R., Rosenberg, A.L., Venkataramani, A.: A Comparison of Dag-Scheduling Strategies for Internet-Based Computing. In: IEEE International Parallel and Distributed Processing Symposium, p. 55 (2007)
6. Schwiegelshohn, U.: An Owner-centric Metric for the Evaluation of Online Job Schedules. In: Proceedings of the 2009 Multidisciplinary International Conference on Scheduling: Theory and Applications, pp. 557–569 (2009)
7. Sinnen, O.: Task Scheduling for Parallel Systems, pp. 108–111. J. Wiley & Sons (2007)
8. 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)
9. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N.: On-line hierarchical job scheduling on grids with admissible allocation. Journal of Scheduling 13(5), 545–552 (2010)
10. Vazquez-Poletti, J.L., Huedo, E., Montero, R.S., Llorente, I.M.: A comparison between two grid scheduling philosophies: EGEE WMS and Grid Way. Journal Multiagent and Grid Systems 3(4), 429–440 (2007)
11. Xhafa, F., Abraham, A. (eds.): Meta-heuristics for Grid Scheduling Problems in Distributed Computing Environments. SCI, vol. 146, pp. 1–37 (2008)

Scheduling Parallel Programs Based on Architecture–Supported Regions

Marek Tudruj[1,2] and Łukasz Maśko[1]

[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
{tudruj,masko}@ipipan.waw.pl

Abstract. Modern multicore processor technology can fairly easily deliver special accelerator processors dedicated to fast optimised execution of critical computational functions. Multi CMP (Chip Multi-Processor) systems can be composed as a set of dedicated and general purpose computational modules interconnected by a global data exchange network. The paper proposes special program scheduling algorithms for such systems. Dedicated CMP modules assumed in the paper are based on a new data communication model called communication on the fly. It enables strong reduction of inter–process and inter–core communication overheads for intensively shared data.

1 Introduction

The use of processor clusters for scientific computations provides improved parallel program execution efficiency and scalability. Multicore processor technology provides new possibilities in the domain of cluster-based systems. The number of cores on a chip is hoped to quickly increase to hundreds and it is the core interconnect design which focuses attention of chip designers rising the interconnect–centric style in the design of Chip MultiProcessors (CMPs) [1, 2]. Technology limitations at the level of internal interconnect fabric and the number of active cores in a CMP make it probable that large monolithic multicore CMP designs will be replaced by hierarchical structures of many technologically sound CMP modules connected by a global network.

In globally interconnected systems of CMP modules a particular strategy in the design of the constituent CMP modules architecture can be applied. Some CMPs can be strongly optimised to provide high parallel speedup for particular types of computations, for example for some program library functions, while other CMPs can be standard multicore processors meant for execution of general purpose code. Usually, the architecturally optimised CMPs are more costly or more difficult to be designed due to some special techniques used, such as more efficient computational facilities or sophisticated interconnection fabrics. Optimal design of program code for so composed system of CMP modules can impose particular requirements on programs. Programs for such executive system can

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 51–60, 2012.

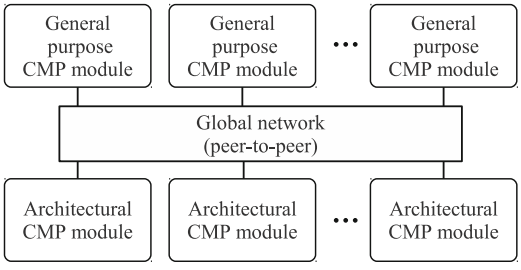


Fig. 1. General system structure

be decomposed into well defined "architecturally supported" regions surrounded by some "glue" code which includes interfaces for these regions.

The paper presents scheduling algorithms for parallel programs which are to be executed in a modular CMP system built of different kinds of constituent CMP modules. The architecture of the specialised CMP modules provides dynamic core switching between CMP shared memory modules (more exactly between L2 data memory busses) and provides data reads on the fly to core L1 caches from L2 memory busses [5–7]. This technique enables efficient direct transmissions of shared data between core L1 data caches inside specialised CMPs. As a result, interconnection structure between cores in specialised CMP modules dynamically adjusts to communication requirements of programs.

Scheduling program tasks in the presence of the architecturally supported regions requires new algorithmic methods and program representation. The paper describes a modified Earliest Task First (ETF) [10] heuristics extended by special task priorities analysis applied to macro data flow graphs of parallel programs. To evaluate the performance of the scheduled exemplary program graphs a simulator is used, which executes structured application program graphs.

The paper is composed of 4 parts. In the first part, the concept of architecturally–supported program regions is explained. In the second part, the proposed architecture of the executive system is described. The third part describes the proposed scheduling algorithm. The fourth part presents experiment results obtained by using the described scheduling algorithms and the program graph execution simulator.

2 Regions in a Program Graph

The algorithm proposed in this paper schedules programs given in a form of macro dataflow graphs. The general structure of the assumed executive parallel multi–CMP system is presented in Fig. 1. Basic system elements are CMP modules interconnected by a global network. There are two kinds of CMP modules: modules with a special architecture, called the architectural CMP modules – ACMPs, and standard modules called general–purpose CMPs – GCMPs (in our case, standard multicore CMP modules). Program graph for such architecture can be logically divided into subgraphs of two forms:

– Subgraphs showing structural or functional features qualifying the respective program parts to be executed in ACMPs which can accelerate their execution. They will be called Architecturally–Supported Regions (ASR). In the case of the ACMPs assumed in this paper, such ASRs should show phase–like regular structure or high level of data sharing, which makes data transfers on the fly and dynamic core switching profitable.
– Subgraphs without promising features for special hardware acceleration. They contain nodes which constitute "a glue" filling the gaps between ASRs and can be executed using a set of general purpose hardware modules with a standard architecture. For the assumed ACMPs, such glue subgraphs have either very irregular (un–phased) structure or they are very loosely data related with low level of data sharing, in which case using core switching and data transfers on the fly will not give performance improvements.

Formally, a program is described by a macro data flow graph $G = (V, E)$, where V, E are the set of nodes and edges of the graph, respectively. G can be divided into two disjoint sets of nodes V_s and V_a, such that $V = V_s \cup V_a$, $V_s \cap V_a = \emptyset$ where V_s contains standard nodes, which constitute a glue, and V_a contains nodes belonging to ASRs. The division may be determined automatically by a compiler, or manually by a programmer. We assume, that incoming data communications to a given ASR may happen only at its beginning (before its computations are started), outgoing data transfers are possible after the region execution is finished and no additional external transfers are allowed. Under such restrictions, ASRs can correspond to subroutines, which may be replaced by a single meta node in the program macro data flow graph. We will call the meta–nodes in such transformed program graph the "architectural nodes".

We assume, that each ASR program subgraph is optimally mapped to cores in an ACMP by separate application of a special scheduling algorithm such as described in [8, 9] for the ACMPs assumed in this paper. We assume that only one ASR subgraph may be executed on a given architectural CMP at a time and that all the cores in this module are potentially enabled for the ASR execution. We also assume, that execution of any ASR may not be interrupted.

3 Architectural CMP Modules with Communication on the Fly

For the experiments performed for this paper we assume that the executive system contains architectural CMP modules supporting data communication on the fly and a set of general purpose multicore shared memory CMP modules. The structure of the assumed architectural CMP module is shown in Fig. 2. Such CMP module contains a number of cores which have private L1 data caches, a number of shared L2 data cache banks, a common shared main memory and a local communication network, which enables connecting L1 caches with L2 data busses. Dynamic core clusters can be created for special group data communication by connecting these core L1 banks to some L2 bank busses. The group

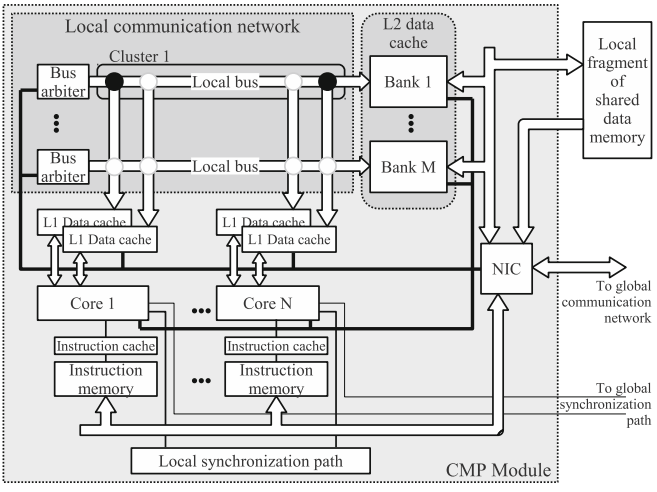


Fig. 2. The structure of an architectural CMP module

communication is oriented on efficiency in using shared data by many cores. Multiple parallel reads of data by many cores to their L1 data caches can be done when some data are present on a L2 cache bus (read on the fly, similar to cache injection [4]). It can be done when a core writes/reads data from/to L1 to/from a L2 bank. The data can be captured to L1 selectively by snooping L2 busses under control of special L1 controllers. A new method for data exchange among dynamic core clusters is supported, called communication on the fly. It consists in dynamic switching of cores with their L1 data cache contents between L2 busses to expose data on the target L2 bus and enable reads on the fly to L1 caches of cores already connected to the L2 bus. Communication on the fly converts standard data transmissions through shared memory and/or some global network, onto dynamic cluster reconfiguration with data transfers performed directly between L1 caches. L1 data caches are multi–ported. It enables parallel pre-fetching of arguments of core n-ary operations, including multiple reads on the fly done in parallel for a core. All data communication and core switching are application program controlled actions. Programs are built of tasks assigned to cores, built according to a cache–controlled macro data–flow paradigm. It requires all data be pre–fetched to core's L1 data cache before a task begins. L1 data cache swapping during tasks is disabled. Computation results can be sent to L2 only between tasks execution. All L2 cache banks of the CMP module are connected to a local fragment of the distributed data main memory shared by all CMP modules in the system. Data communication between fragments of the CMP shared memory is possible under supervision of NICs (Network Interface Controllers). Reads on the fly to L2 modules during data transfers among CMP shared memory fragments are also possible. More details on the proposed architecture of the architectural CMP modules and communication control, can be found in [5–7].

4 Task Scheduling with Architecturally Supported Regions

The presented algorithm aims at minimal program execution time by obtaining equal load of ACMP and GCMP modules. It schedules glue nodes to GCMPs and architectural subgraphs to ACMPs. The algorithm is based on list scheduling with the ETF heuristics [10], but the node selection is extended by classification of nodes in the program graph to obtain gradual load balancing of both general purpose and architectural CMPs. The nodes are classified by priorities, which at each point of the scheduling algorithm allow selection of only such glue nodes, whose results are required for execution of the topologically nearest ASRs in the graph. Other standard nodes are scheduled within the remaining available processing resources without delaying higher classified nodes.

The architectural nodes are divided into layers by assignment of the 1^{st} level priorities (denoted as $pr_1(v)$ for each node v). Each such layer contains a subset of pair–wise independent nodes, i.e. with no data dependencies between any two of them. The nodes for layers are selected according to topological properties of the graph, using graph paths analysis. Such layers are then scheduled one–by–one. The 1^{st} level priorities are computed using an "architectural task graph" $G_a = (V_a, E')$. In this graph, nodes correspond to architectural nodes in an initial graph G (the V_a set). For two nodes $u, v \in V_a$, an edge $u \rightarrow v \in E'$ exists in G_a, if there is a directed path between these two nodes in the original graph G containing only glue nodes. For each $u \in V_a$, $pr_1(u)$ is equal to its depth in G_a (the number of nodes on the longest path leading to u from one of the

Algorithm 1. Definition of 2^{nd} level priorities for a set U of nodes

Let $p = 0$ be the lowest priority value for a set U of graph nodes.
while U is not empty **do**
  Find X_u sets for all nodes from U. Let $V = \emptyset$ and $X_V = \emptyset$.
  while $|V|$ is smaller than the number of ACMPs **do**
    if V is empty **then**
      for all tasks u from U **do**
        Schedule a subgraph X_u on available cores inside GCMPs, using an ETF–based list scheduling.
      end for
      Select such node u from U, for which its X_u set gives the shortest schedule in the previous step and uses the smallest number of resources.
    else
      Select node $u \in U$ such, that $X_u \cap X_V$ is the biggest.
    end if
    Let $V = V \cup \{u\}$ and $X_V = X_V \cup X_u$
  end while
  Assign $pr_2(u) = p$ for all nodes u from $V \cup X_V$.
  Remove the architectural nodes, which belong to V from U.
  Let $p = p + 1$.
end while

nodes which have no predecessors in G_a). Priorities for glue nodes are derived from priorities of architectural nodes. For each $v \in V_s$, a set $X \subset V_a$ of nodes is determined such that there exists a path from node v to each of these nodes. If X is not empty, the priority of v is equal to $min_{u \in X}(pr_1(u))$, and it is equal to $\max_{u \in V_a}(pr_1(u)) + 1$ otherwise. A set of nodes with the same 1^{st} level priority constitutes a layer of nodes.

The 2^{nd} level priority (denoted as $pr_2(v)$ for each node v) aims at division of nodes for layers determined by the 1^{st} level priority, into subsets in such way that we can obtain equal, high load of all CMPs. Each subset of nodes contains no more architectural nodes than the number of ACMPs in the system. We determine 2^{nd} level priorities, if there is a set U of architectural nodes, which have the same 1^{st} level priorities. For each node $u \in U$ we determine X_u as a set of standard nodes $v \in V$ such that there exists a directed path from v to u, $pr_1(v) = pr_1(u)$, and they have no 2^{nd} level priority assigned yet. The nodes in such sets are used to put architectural nodes in order. Priorities for nodes from the U–sets are assigned using the algorithm shown as Algorithm 1.

The final priority $pr(v)$ is defined as follows: for two nodes u and v (both must be either architectural or standard), $pr(u) < pr(v) \iff pr_1(u) < pr_1(v) \vee (pr_1(u) = pr_1(v) \wedge pr_2(u) < pr_2(v))$.

Algorithm 2. List scheduling algorithm with modified ETF heuristics

1: {Input: a program graph $G = (V, E)$}
2: Determine architectural task graph G' based on graph G and, based on it, compute priorities $pr_1(v)$ for all nodes $v \in G$.
3: **for** all nodes $v \in G$ determine $pr_2(v)$ using Algorithm 1 **do**
4: Let P be the set of ready nodes from graph G. Initially, insert all the nodes without predecessors from G into P.
5: **while** P is not empty **do**
6: Let $pr_{min} = \min_{u \in P}(pr(u))$
7: **for** all nodes $u \in P$ such that $pr(u) = pr_{min}$ **do**
8: Select the node u with the earliest possible execution start time. Let p be the core, on which this execution is available.
9: **end for**
10: **for** all nodes $v \in P$ such that $pr(v) > pr_{min}$ **do**
11: If possible, select the node v with earliest possible execution start time and for which execution ends before execution of the node u selected in the previous loop may be started. Let q be the core, on which such execution is possible.
12: **end for**
13: **if** the node v has been selected **then**
14: Let $u = v$ and $p = q$
15: **end if**
16: Schedule the node u for execution on the core p and remove it from P.
17: Insert to P all the descendants of the node u, for which all their predecessors have already been scheduled.
18: **end while**
19: **end for**

The scheduling algorithm (Algorithm 2) is based on list scheduling. Glue nodes are scheduled for execution on GCMPs, architectural nodes are scheduled for execution on ACMPs. Each time, when a node is to be selected, first, the nodes with the lowest priorities are examined and scheduled. Other nodes are scheduled only when their execution doesn't interfere with execution of those which have currently the lowest priority. Such selection of nodes leads to a situation, where architectural nodes may be executed as soon as possible, using cores from architectural CMPs. In the same time, general purpose cores may be used to execute standard nodes, whose execution doesn't depend on architectural nodes and which are required for further computations.

5 A Scheduling Example

As an example, scheduling of a program graph of form shown in Fig. 3 is used. This graph consists of a set of nodes which compute a value of a functions Fi, Gi and multiplication (Mi) on square matrices. The Fi and Gi nodes have irregular internal structure, but can be divided into independent nodes of a smaller size. Each node is performed on square matrices of a given size, which is a parameter.

Each of matrix multiplication nodes M1, . . . ,Mn from Fig. 3a is expanded using parallel Strassen matrix multiplication method. Such graph unrolled at 1^{st} recursion level is presented in Fig. 3b. Also nodes Fi and Gi were parallelised as shown in Fig. 3c and 3d to match the matrix sizes imposed by final serial multiplication operations in the graph. In the unrolled graph, all the Fn^i, Gn^i and addition nodes An^i are treated as glue, only multiplications Mn^i inside subgraphs from Fig. 3b are implemented as architectural nodes and executed inside ACMPs. Inside an architectural CMP module, each multiplication node is parallelised using decomposition of matrices into quarters and structured to use core switching and reads on the fly. For our experiments we have limited the depth of the graph in Fig. 3a to 3 multiplication nodes ($n = 3$).

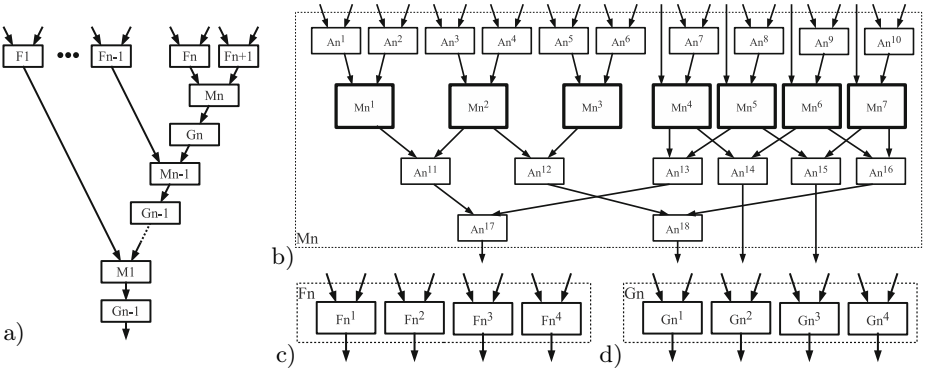


Fig. 3. The general view of the exemplary program graph (a), graph of the Mn node following parallel Strassen method (b) and graphs for node Fn (c) and Gn (d)

Table 1. Parallel speedup of the scheduled program graph

Rec. level	System config. ACMPs	GCMPs	Speedup over	Matrix size 32	64	128	256	512
1	7	1	serial	24,402	29,195	35,531	37,525	44,936
			ETF	1,130	1,195	1,248	1,331	1,302
		2	serial	24,384	29,643	34,128	39,494	43,912
			ETF	1,115	1,105	1,149	1,167	1,197
		8	serial	32,470	39,335	47,575	53,522	60,607
			ETF	1,013	1,018	1,026	1,073	1,060
2	49	1	serial	24,002	53,387	61,505	72,642	60,822
			ETF	1,052	1,054	1,044	1,013	1,070
		4	serial	45,116	99,689	117,146	141,822	153,406
			ETF	1,142	1,157	1,200	1,262	1,270
		8	serial	52,199	114,818	142,462	174,640	197,307
			ETF	1,056	1,118	1,117	1,138	1,197
		32	serial	56,494	132,290	168,018	219,307	265,135
			ETF	1,006	1,008	1,012	1,017	1,080
3	343	2	serial	20,958	46,188	107,499	137,845	181,543
			ETF	1,000	1,000	1,014	1,055	1,115
		4	serial	41,235	90,864	210,670	260,937	330,366
			ETF	1,043	1,026	1,042	1,089	1,113
		8	serial	77,976	171,032	382,424	452,388	562,933
			ETF	1,182	1,139	1,105	1,103	1,058
		16	serial	114,037	255,967	570,518	728,020	959,998
			ETF	1,136	1,163	1,150	1,213	1,287
		32	serial	124,540	279,513	656,417	834,289	1044,773
			ETF	1,013	1,039	1,137	1,129	1,151
		128	serial	129,250	289,167	681,178	883,506	1196,697
			ETF	1,004	1,006	1,006	1,009	1,014

If such graph is scheduled using a classic ETF list scheduling without priorities, the F-nodes are examined and scheduled in the order as in Fig. 3a ($F1^i$ – $Fn{+}1^i$). The critical path of this graph consists of multiplication nodes, $Fn{+}1^i$ and Gi^i nodes, therefore such scheduling leads to unoptimal execution. Optimal schedule should first execute nodes Fn^i and $Fn{+}1^i$, followed by addition nodes An^i, enabling execution of the nodes Mn^i. While these nodes are executed using ACMPs, other F-nodes may be executed in parallel by GCMPs, thus providing the best execution time. Experiments have shown, that the presented algorithm with priorities leads to such schedule. In all the experiments a software simulator of graph execution in the proposed architecture was used.

Table 1 shows parallel speedup of the scheduled graph for 3 different recursion levels, different matrix sizes and different system configurations, compared to standard serial multiplication and the schedule obtained using a classical ETF list scheduling without priorities. The numbers of ACMPs and GCMPs (8 and 4 cores, respectively) were so selected as to enable full parallelization of multiplication in ACMPs and to check the influence of the number of GCMPs on

the overall program performance due to potential serialization of the execution of the glue. Cores in ACMPs and GCMPs are assumed to be equally fast in terms of computation. Computations were assumed to be 3 times faster than local communication on the L1–L2 bus. Global communication was assumed to be 4 times slower than the local one.

The number of ACMPs was assumed to match the parallel multiplication graph width for each recursion level. If the number of GCMPs is small, execution of architectural regions of the graph can be forced to wait for the glue to be computed, which slows down execution. Increasing the number of GCMP modules improves parallel execution. Speedup increases up to 8 GCMPs at recursion level 1 (32 and 128 GCMPs at rec. levels 2 and 3, respectively) and then it saturates.

For a fixed matrix size, a higher recursion level means larger graphs and smaller computation grain. It means, that the parallel execution of the graph requires more resources (both ACMP and GCMP modules) to obtain good performance. If the executive system contains enough GCMP modules, a higher recursion level leads to higher overall speedup, due to greater use of parallelism.

Increasing the size of a matrix leads to a bigger parallel computation grain. For considered matrix sizes (32–512), the computation grain is determined by the size of the matrix parts used for elementary serial multiplication after recursive decomposition at a given recursion level, at the 3^{rd} recursion level, the grain is the smallest: 2, 4, 8, 16 and 32, respectively. A bigger grain – in most cases – gives better parallel speedup, but we can observe departure from this rule at recursion level 1 with 2 GCMP modules as well as at recursion level 2 with 1 and 2 GCMP modules.

Introduction of priorities in almost all cases improves execution, comparing to a standard ETF-based list scheduling algorithm. We can observe the best improvements for the highest size of matrices (512). For recursion level 1, the presented algorithm improved parallel execution up to 1.3 times (for 1 GCMP module), for rec. level 2 the best improvement was 1.27 (for 4 GCMP modules) and for rec. level 3 it was 1.28 (for 16 GCMP modules). Apart from the situation, when the number of GCMPs is definitely too small for execution of a big parallel graph (1 GCMP for recursion levels 2 and 3), the positive influence of the special priorities is especially visible when the execution of the glue is "squeezed" to a small number of GCMPs.

6 Conclusions

The paper has presented parallel program scheduling algorithms for modular CMP systems composed of two kinds of CMP modules interconnected by a global network: architecturally supported CMP modules — ACMPs and general purpose shared memory CMP modules — GCMPs. The proposed scheduling algorithm is based on an ETF heuristics improved by insertion of multi–level priorities for the nodes in application program graphs. The additional priorities enable better synchronisation of execution of architecturally supported regions and so called "glue" nodes in application programs. Thus, better parallel

speedups were obtained in the case of graph structures "difficult" for the standard ETF schedulers for adequate composition of the executive system. The proposed scheduling algorithm was experimentally verified on a graph of matrix computation in which matrix multiplications by Strassen method were architecturally supported for parallel execution of Strassen constituent submatrix multiplications by matrix decomposition into quarters. To obtain maximal parallel speedup, the number of CMP modules should match the width of the program graph: the number of ACMPs should match the Strassen method recursion level and the number of GCMPs should match the width of the glue regions of the graph.

References

1. Nurmi, J., Tenhunen, H., Isoaho, J., Jantsch, A. (eds.): Interconnect–Centric Design for Advanced SOC and NOC. Springer (2004)
2. Kundu, S., Peh, L.S.: On–Chip Interconnects for Multicores. IEEE Micro, 3–5 (September-October 2007)
3. Owens, J.D., et al.: Research Challenges for On–Chip Interconnection Networks. IEEE Micro, 96–108 (September-October 2007)
4. Milenkovic, A., Milutinovic, V.: Cache Injection: A Novel Technique for Tolerating Memory Latency in Bus-Based SMPs. In: Bode, A., Ludwig, T., Karl, W.C., Wismüller, R. (eds.) Euro-Par 2000. LNCS, vol. 1900, pp. 558–566. Springer, Heidelberg (2000)
5. Tudruj, M., Maśko, Ł.: Dynamic SMP Clusters with Communication on the Fly in NoC Technology for Very Fine Grain Computations. In: 3rd Int. Symp. on Parallel and Distributed Computing, ISPDC 2004, Cork, pp. 97–104 (July 2004)
6. Tudruj, M., Maśko, Ł.: Towards Massively Parallel Computations Based on Dynamic SMP Clusters wih Communication on the Fly. In: Proceedings of the 4th International Symposium on Parallel and Distributed Computing, ISPDC 2005, Lille, France, July 4-6, pp. 155–162. IEEE CS Press (2005)
7. Tudruj, M., Maśko, Ł.: Data Transfers on the Fly for Hierarchical Systems of Chip Multi–Processors. In: Wyrzykowski, R., et al. (eds.) PPAM 2011, Part I. LNCS, vol. 7203, pp. 50–59. Springer, Heidelberg (2012)
8. Masko, 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)
9. Maśko, Ł., Tudruj, M.: Task Scheduling for SoC–Based Dynamic SMP Clusters with Communication on the Fly. In: 7th Int. Symp. on Parallel and Distributed Computing, ISPDC 2008, pp. 99–106 (2008)
10. 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)

Genetic Algorithm Calibration for Two Objective Scheduling Parallel Jobs on Hierarchical Grids

Victor Hugo Yaurima-Basaldua[1], Andrei Tchernykh[2], Yair Castro-Garcia[3], Victor Manuel Villagomez-Ramos[1], and Larisa Burtseva[3]

[1] CESUES Superior Studies Center, San Luis R.C., Mexico
vyaurima@yahoo.com, vicmvr@msn.com
[2] CICESE Research Center, Ensenada, Mexico
chernykh@cicese.mx
[3] Autonomous University of Baja California, Mexicali, Mexico
yair.castro.garcia@gmail.com, lpb@iing.mxl.uabc.mx

Abstract. This paper addresses non-preemptive offline scheduling parallel jobs on a Grid. We consider a Grid scheduling model with two stages. At the first stage, jobs are allocated to a suitable Grid site, while at the second stage, local scheduling is independently applied to each site. In this environment, one of the big challenges is to provide a job allocation that allows more efficient use of resources and user satisfaction. In general, the criteria that help achieve these goals are often in conflict. To solve this problem, two-objective genetic algorithm is proposed. We conduct comparative analysis of five crossover and three mutation operators, and determine most influential parameters and operators. To this end multi factorial analysis of variance is applied.

Keywords: Offline scheduling, Grid, Genetic Algorithm, Crossover Operator, Mutation Operator.

1 Introduction

In this paper, we present experimental work in the field of multi-objective scheduling in a two layer Grid computing. At the first layer, we select the best suitable machine for each job using a given criterion. At the second layer, local scheduling algorithm is applied to each machine independently. In such an environment, one of the big challenges is to provide scheduling that allows more efficient use of resources, and satisfies other demands. The optimization criteria are often in conflict. For instance, resource providers and users have different objectives: providers strive for high utilization of resources, while users are interested in a fast response. We provide solutions that consider both goals. The aggregation method of criteria and a scalar function to normalize them are used. We examine the overall Grid performance based on real data and present a comprehensive comparative analysis of five crossover operators, three operators of

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 61–70, 2012.

mutation, five values of crossover probability, and five values of mutation probability. To genetic algorithm tune up the multifactorial analysis of variance is applied.

After formally presenting our Grid scheduling model in Section 2, we discuss related work in Section 3. We introduce genetic scheduling algorithms and discuss their application for Grid scheduling in Section 4. Genetic algorithm calibration is presented in section 5. Finally, we conclude with the summary in Section 6.

2 Model

We address an offline scheduling problem: n parallel jobs $J_1, J_2, ..., J_n$ must be scheduled on m parallel machines (sites) $N_1, N_2, ..., N_m$. Let m_i be the number of identical processors or cores of machine N_i. Assume without loss of generality that machines are arranged in non-descending order of their numbers of processors, that is $m_1 \leq m_2 \leq ... \leq m_m$ holds.

Each job J_i is described by a tuple $(size_j, p_j, p_j^{'})$: its size $1 \leq size_j \leq m_m$ also called processor requirement or degree of parallelism, execution time p_j and user runtime estimate $p_j^{'}$. The release date of a job is zero; all the jobs are available before scheduling process start. Job processing time is unknown until the job has completed its execution (non-clairvoyant case). User runtime estimate $p_j^{'}$ is provided by a user. A machine must execute a job by exclusively allocating exactly $size_j$ processors for an uninterrupted period of time p_j to it. As we do not allow multisite execution and co-allocation of processors from different machines, a job J_j can only run on machine N_i if $size \leq m_j$ holds.

Two criteria are considered: *makespan*: C_{max}, $C_{max} = max(C_i)$, where C_i is the maximum completion time on N_i machine $(i = 1, 2, 3, ..., N_m)$ and *mean turnaround time*: $TA = \frac{1}{n}\sum_{j=1}^{n} c_j$, where c_j is the completion time of job J_j.

We denote our Grid model by GP_m. In the three field notation $(\alpha\,|\beta|\,\gamma)$ introduced in [6], our scheduling problem is characterized as $GP_m \left| size_j, p_j, p_j^{'} \right| OWA$, where OWA is the value of the multi-criteria aggregation operator ($OWA = w_1 C_{max} + w_2 TA$), and w_i is the linear combination weights. The problem of scheduling on the second stage is denoted as $P_m \left| size_j, p_j, p_j^{'} \right| C_{max}$.

3 Related Work

3.1 Hierarchical Scheduling

Scheduling algorithms for two layer Grid models can be split into a global allocation part and a local scheduling part. Hence, we regard MPS (Multiple machine Parallel Scheduling) as a two stage scheduling strategy: $MPS = MPS\_Alloc + PS$ [20]. At the first stage, we allocate a suitable machine for each job using a genetic algorithm. At the second stage, PS (single machine Parallel Scheduling) algorithm is applied to each machine independently for jobs allocated during the previous stage.

3.2 Multi-criteria Scheduling

Several studies deal with scheduling in Grids considering only one criterion (e.g. EGEE Workload Management System [1], NorduGrid broker [4], eNANOS [16], Gridway [10]. Various Grid resource managements involve multiple objectives and may use multicriteria decision support. Dutot et al. [3] considered task scheduling on the resources taking into account maximum task completion time and the average completion time. Siddiqui, et al. [18] presented task scheduling based on the resource negotiation, advance reservation, and user preferences. The user preferences are modeled by utility functions, in which users must enter the values and levels of negotiation.

General multi-criteria decision methodology based on the Pareto optimality can be applied. However, it is very difficult to achieve fast solutions needed for Grid resource management by using the Pareto dominance. The problem is very often simplified to a single objective problem or objectives' combining. There are various ways to model preferences, for instance, they can be given explicitly by stakeholders to specify an importance of every criterion or a relative importance between criteria. This can be done by a definition of criteria weights or criteria ranking by their importance.

In order to provide effective guidance in choosing the best strategy, Ramirez et al. [15] performed a joint analysis of several metrics according to methodology proposed in [19]. They introduce an approach to multi-criteria analysis assuming equal importance of each metric. The goal is to find a robust and well performing strategy under all test cases, with the expectation that it will also perform well under other conditions, e.g., with different Grid configurations and workloads. Kurowski et al. [11] used aggregation criteria method for modeling the preferences of participants (owners, system managers and users). Authors considered two stage hierarchical grids, taking into account the stakeholders' preferences, assuming unknown processing times of the tasks, and studied the impact of the size of the batch of tasks on the efficiency of schedules. Lorpunmanee et al. [12] presented task allocation strategies to the different sites of a Grid and propose a model for task scheduling considering multiple criteria. They concluded that such scheduling can be performed efficiently using *GAs*.

In this paper, we consider two-criteria scheduling problem. We propose a genetic algorithm as a strategy for allocating jobs to resources. It uses an aggregation criteria method and the weight generating function representing the relative importance of each criterion. We present an experimental analysis of such a problem and compare obtained results with strategies aimed at optimizing a single criterion. In this paper, the Ordered Weighted Averaging (OWA) [11] is applied: $OWA(x_1, x_2, ..., x_n) = \sum_{c=1}^{k} w_c s(x)_{\sigma(c)}$, where w_c is the weight, $c = 1, ..., k$, x_c is a value associated with the satisfaction of the c criterion. Permutation ordering values: $s(x)_{\sigma(1)} \leq s(x)_{\sigma(2)} \leq ...s(x)_{\sigma(k)}$ is performed. Weights (w_c) are nonnegative and $\sum_{c=1}^{k} w_c = 1$. If all the weights are set to the same value, OWA behaves as the arithmetic mean. In this case, high values of some criterion compensate low values of the other ones. In OWA approach, a compromise solution is provided. The highest weight is w_1 and the subsequent ones are

decreasing, but never reaching 0. That means that both the worst criterion and the mean of criteria are taken into account. In this way stakeholders are able to evaluate schedules using multiple criteria. The goal is to find a weighting scheme that provides the best possible mean value for all stakeholders' evaluations and the highest possible value for the worst case evaluation. To achieve this, weight w_1 must be relatively large, while weight w_k should be small, where k denotes the number of criteria. The remaining weights decrease in value from w_1 to w_k according to:

$$w_c = \begin{cases} 3/2k, & c = 1 \\ (3k - 2c - 1)/\left[2n(k-1)\right], & 1 < c \leq k \end{cases} \tag{1}$$

4 Genetic Algorithm

Scheduling algorithms for two layer Grid models can be split into a global allocation part and a local scheduling part. Hence, we regard *MPS* as a two stage scheduling strategy: *MPS*=*MPS_Alloc*+*PS*. At the first stage, we use a genetic algorithm (*GA*) to allocate a suitable machine for each job (*MPS_Alloc*=*GA*). At the second stage, the parallel job scheduling algorithm *PS* is applied to each machine independently for jobs allocated during the previous stage. As *PS* we use the well-known strategy Backfilling-EASY [21,22].

GA is a well-known search technique used to find solutions to optimization problems [9]. Candidate solutions are encoded by chromosomes (also called genomes or individuals). The set of initial individuals forms the population. Fitness values are defined over the individuals and measures the quality of the represented solution. The genomes are evolved through the genetic operators generation by generation to find optimal or near-optimal solutions. Three genetic operators are repeatedly applied: selection, crossover, and mutation. The selection picks chromosomes to mate and produce offspring. The crossover combines two selected chromosomes to generate next generation individuals. The mutation reorganizes the structure of genes in a chromosome randomly so that a new combination of genes may appear in the next generation. The individuals are evolved until some stopping criterion is met. OWA operator as a fitness function is applied (Section 2).

Each solution is encoded by $n \cdot m$ matrix. Where m is a number of machines in a Grid, and n is a number of jobs. The $i = 0, ..., m-1$ row represents local queue of machine N_i. Note, that machines are arranged in non-descending order of their number of processors $m_0 \leq m_2 \leq m_{m-1}$. A job J_j can only run on machine N_i if $size_j \leq m_i$ holds. The available set of machines for a job J_j is defined to be the machines with indexes $f_j, ..., m$, where f_j is the smallest index i such that $m_i \geq size_j$.

The selection picks chromosomes to produce offspring. The binary tournament selection known as an effective variant of the parents' selection is considered. Two individuals are drawn randomly from the population, and one with highest fitness value wins the tournament. This process is repeated twice in order to select two parents.

4.1 Crossover Operators

Crossover operator produces new solutions by combining existing ones. It takes parts of solution encodings from two existing solutions (parents) and combines them into single solution (child). The crossover operator is applied under a certain probability (P_c). In this paper, five operators are considered.

One Segment Crossover for Matrix ($OSXM$). It is based on the crossover operator *OSX - One Segment Crossover* [7]. In this crossover, two random points $S1$ and $S2$ from 0 to $vmax$ (maximum index used) are selected. The child inherits columns from the 0 to $S1$ position from parent 1. It also inherits columns from $S1$ to $S2$ from parent 2, but only elements that have not been copied from parent 1. Finally, child inherits elements from parent 1 that have not yet been copied.

Two Point Crossover for Matrix (TPM). It is based on the crossover operator *Two Point Crossover* [13]. In this crossover, two random points $S1$ and $S2$ from 0 to $vmax$ are selected. Columns from position 0 to $S1$ and from $S2$ to $vmax$ are copied from parent 1. The remaining elements are copied from the parent 2 only if they have not been copied.

Order Based Crossover for Matrix ($OBXM$). It is based on the crossover operator *OBX - Order Based Crossover* [5]. A binary mask is used. The mask values equal to one indicate that the corresponding columns are copied from parent 1 to the child. The rest of elements are copied from parent 2, only if they have not been copied. The mask values are generated randomly and uniformly.

Precedence Preservative Crossover for Matrix ($PPXM$). It is based on the crossover operator *PPX - Precedence Preservative Crossover* [2]. Random binary mask values equal to one indicates that corresponding columns are copied from parent 1 to the child, and the values equal to zero indicate that columns are copied from parent 2, this is done by iterating the columns of parents who have not been copied in order from left to right.

Order Segment Crossover for Matrix with Setup ($OSXMS$). It is based on the crossover operator *OSX - Order Segment Crossover* [7]. It chooses two points randomly. Columns from position 1 to the first point are copied from parent 1. The elements are ordered by the number of processors required for subsequent insertion into the child in the position according to the number of required processors. Columns from first point to second point are copied from parent 2, only if the elements have not been copied. Finally, the remaining elements are copied from the parent 1, considering not copied elements.

4.2 Mutation

The mutation operator produces small changes in an offspring with probability P_m. It prevents falling of all solutions into local optimum and extends search space of the algorithm. Three operators *Insert*, *Swap* and *Switch* adapted for two-dimensional encoding are considered. 1)Queue_Insert. Two points are randomly selected. The element of the second point is inserted to the first point, shifting the rest. Note that this mutation preserves most of the order and the

adjacency information. 2)Queue_Swap. This mutation randomly selects two points and swaps their elements. 3)Queue_Switch. This mutation selects a random column and swaps their elements with the next column.

5 GA Calibration

5.1 Workload

The accuracy of the evaluation highly relies upon workloads applied. For testing the job execution performance under a dedicated Grid environment, we use Grid workload based on real production traces. Carefully reconstructed traces from real supercomputers provide a very realistic job stream for simulation-based performance evaluation of Grid job scheduling algorithms. Background workload (locally generated jobs) that is an important issue in non-dedicated Grid environment is not addressed.

Four logs from PWA (Parallel Workloads Archive) [14] (Cornell Theory Center, High Performance Computing Center North, Swedish Royal Institute of Technology and Los Alamos National Lab) and one from GWA (Grid Workloads Archive) [8] (Advanced School for Computing and Imaging) have been used: The archives contain real workload traces from different machine installations. They provide information of individual jobs including submission time, and resource requirements.

For creating suitable grid scenarios, we integrate several logs by merging users and their jobs. The premise for the integration of several logs of machines in production use into a Grid log is based on the following. Grid logs contain jobs submitted by users of different sites; a Grid execution context could be composed by these sites. Unification of these sites into a Grid will trigger to merger users and their jobs. It should be mentioned that merging several independent logs to simulate a computational Grid workload does not guarantee representation of the real Grid with the same machines and users. Nevertheless, it is a good starting point to evaluate Grid scheduling strategies based on real logs in the case of the lack of publicly available Grid workloads. Time zone normalization, profiled time intervals normalization, and invalid jobs filtering are considered.

5.2 Calibration Parameters

A method of experimental design is adapted from Ruiz and Maroto [17], where the following steps are defined: (a) test all instances produced with possible combinations of parameters; (b) obtain the best solution for each instance; (c) apply the Multifactor Variance Analysis (ANOVA) with 95% confidence level to find the most influential parameters; (d) set algorithm parameters based on selected parameters values; (e) calculate relative difference of the calibrated algorithm and other adapted algorithms over the best solutions. Table 1 shows parameters that were set for the calibration. Hence, 5 x 3 x 5 x 5 = 375 different algorithms alternatives were considered. 30 executions of the workload were realized, in total 375 x 30 = 11,250 experiments.

Table 1. Calibration parameters

Instance	Levels
Crossover operators	OSXM, TPM, OBXM, PPXM, OSXMS
Mutation operators	Queue_Insert, Queue_Swap, Queue_Switch
Crossover probability	0.001, 0.01, 0.05, 0.1, 0.2
Population	50 individuals
Number of jobs in individual	5275
Selection	Binary tournament
Stop criterion	If the fitness value of the best chromosome is not improved 4 times, the algorithm is stopped.

The performance of the proposed algorithm is calculated as the percentage of the relative distance of the obtained solution from the best one (Relative Increment over the Best Solution - *RIBS*). The *RIBS* is calculated using the following formula: $RIBS = (Heu_{sol} - Best_{sol})/Best_{sol} \cdot 100$, where Heu_{sol} is the value of the objective function obtained by considered algorithm, and $Best_{sol}$ is the best value obtained during the testing all possible parameter combinations.

5.3 Analysis of Variance

To assess the statistical difference among the experimental results, and observe effect of different parameters on the result quality, the ANOVA is applied. The analysis of variance is used to determine factors that have a significant effect, and which are the most important factors. Parameters of the Grid scheduling problem are considered as factors, and their values as levels. We assume that there is no interaction between the factors.

Table 2. Analysis of variance for *RIBS* (Type III Sums of Squares)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:cross	2111.03	4	527.758	152.00	0.0000
B:mut	47.4724	2	23.7362	6.84	0.0012
C:pc	20.9586	4	5.23964	1.51	0.1990
D:pm	27.5114	4	6.87785	1.98	0.0969
RESIDUAL	1249.99	360	3.4722		
TOTAL (CORRECTED)	3456.97	374			

The *F-Ratio* is the ratio between mean square of the factor and the mean square of residues. A high *F-Ratio* means that this factor affects the response variable (see Table 2). The value of *P-Value* shows the statistical significance of the factors. The factors, whose *P-Value* is less than 0.05, have statistically significant effect on the response variable (*RIBS*) with 95% level of confidence.

Here we see that the factors that significantly affect the response variable are the operators of crossover and mutation. According to the *F-Ratio*, the most important factor is the crossover operator.

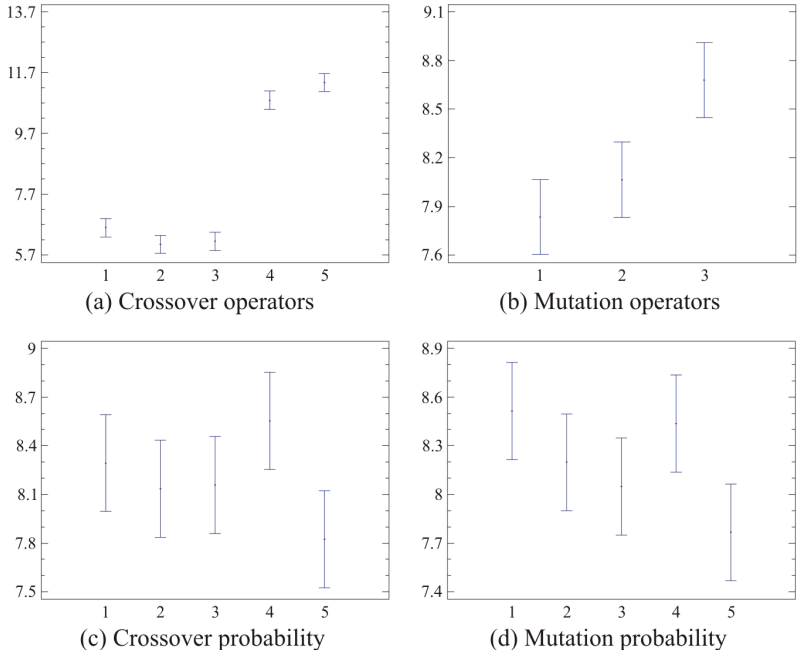


Fig. 1. Means and 95% LSD intervals

Figs. 1(a)-1(d) show means and 95% LSD intervals of the most influential factors. Fig. 1(a) shows the results obtained for crossover operators. Five operators are presented in the following order: 1-*OSXM*, 2-*TPM*, 3-*OBXM*, 4-*PPXM* and 5-*OSXMS*. The vertical axis is the values of *RIBS*. We can see that crossover *TPM* is the best crossover among the five ones tested, followed by *OBXM*. Figure 1(b) shows the results obtained for mutation operators. Three mutation operators are presented in the following order: 1- *Queue_Insert*, 2- *Queue_Swap* y 3- *Queue_Switch*. *Queue_Insert* is shown to be the best, followed by *Queue_Swap*. Figures 1(c) and 1(d) show the results for the crossover and mutation probability. The best crossover probability occurring is 0.9. The best mutation probability occurring is 0.2.

6 Conclusions and Future Work

We addressed a two-objective genetic algorithm calibration for scheduling jobs in a two stage computational Grid. We conduct comprehensive comparison of

five known crossover operators, three mutation operators adapted for two dimension encoding, five values for the crossover probability, and five values for the mutation probability. 375 different genetic algorithms are analysed with 30 instances for each one. 11,250 experiments were evaluated. We use aggregation criteria method for modeling the preferences of two objectives: C_{max} and TA. To assess the statistical difference among the experimental results, and observe impact of different parameters on the result quality, the ANOVA technique was applied. The crossover operator is shown to have a statistically significant effect. It plays the most important role in genetic algorithms applied to a scheduling in computational Grid. Of five compared operators, the *TPM* obtained the best result, followed by the *OBXM*.

Obtained results may serve as a starting point for future heuristic Grid scheduling algorithms that can be implemented in real computational Grids. While the scope of this work is to determine most influential *GA* parameters and operators, in future work, we also intend to evaluate the practical performance of the proposed strategies, and the assessment of their cost. To this end, we plan simulations using real workload traces and corresponding Grid configurations. We will compare our *GA* with other existing Grid scheduling strategies which are typically based on heuristics. Future work needs for a better understanding of the scheduling with dynamic Grid configuration, resource failures and other real characteristics of the Grid.

References

1. Avellino, G., Beco, S., Cantalupo, B., Maraschini, A., Pacini, F., Terracina, A., Barale, S., Guarise, A., Werbrouck, A., Di Torino, S., Colling, D., Giacomini, F., Ronchieri, E., Gianelle, A., Peluso, R., Sgaravatto, M., Mezzadri, M., Prelz, F., Salconi, L.: The EU datagrid workload management system: towards the second major release. In: 2003 Conference for Computing in High Energy and Nuclear Physics. University of California, La Jolla (2003)
2. Bierwirth, C., Mattfeld, D., Kopfer, H.: On Permutation Representations for Scheduling Problems. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996, Part II. LNCS, vol. 1141, pp. 310–318. Springer, Heidelberg (1996)
3. Dutot, P., Eyraud, L., Mounie, G., Trystram, D.: Models for scheduling on large scale platforms: which policy for which application? In: Proceedings of 18th International Symposium on Parallel and Distributed Processing, p. 172 (2004)
4. Elmroth, E., Tordsson, J.: An interoperable, standards based Grid resource broker and job submission service. In: First International Conference on e-Science and Grid Computing, 2005, pp. 212–220. IEEE Computer Society, Melbourne (2005)
5. Gen, M., Cheng, R.: Genetic algorithms and engineering optimization, p. 512. John Wiley and Sons, New York (1997)
6. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P.L., Johnson, E.L., Korte, B.H. (eds.) Discrete Optimization II. Annals of Discrete Mathematics, vol. 5, pp. 287–326. North-Holland, Amsterdam (1979)

7. Guinet, A., Solomon, M.: Scheduling Hybrid Flowshops to Minimize Maximum Tardiness or Maximum Completion Time. Int. J. Production Research 34(6), 1643–1654 (1996)
8. GWA Grid Workloads Archive, http://gwa.ewi.tudelft.nl
9. Holland, J.: Adaptation in natural and artificial systems. University of Michigan Press (1975)
10. Huedo, E., Montero, R.S., Llorente, I.M.: Evaluating the reliability of computational grids from the end user's point of view. Journal of Systems Architecture 52(12), 727–736 (2006)
11. Kurowski, K., Nabrzyski, J., Oleksiak, A., Wglarz, J.: A multicriteria approach to two-level hierarchy scheduling in Grids. Journal of Scheduling 11(5), 371–379 (2008)
12. Lorpunmanee, S., Noor, M., Hanan, A., Srinoy, S.: Genetic algorithm in Grid scheduling with multiple objectives. In: Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, pp. 429–435 (2006)
13. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolutions Programs, 3rd edn. Springer, Heidelberg (1996)
14. PWA Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/
15. Ramirez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., Gonzalez-Garcia, J.-L., Hirales-Carbajal, A.: Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids. J. Grid Computing 9, 95–116 (2011)
16. Rodero, I., Corbalán, J., Badía, R.M., Labarta, J.: eNANOS Grid Resource Broker. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 111–121. Springer, Heidelberg (2005)
17. Ruiz, R., Maroto, C.: A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research 169, 781–800 (2006)
18. Siddiqui, M., Villazon, A., Fahringer, T.: Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC 2006). ACM, New York, article 103 (2006), http://doi.acm.org/10.1145/1188455.1188563
19. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. 18, 789–803 (2007)
20. Tchernykh, A., Schwiegelsohn, U., Yahyapour, R., Kuzjurin, N.: Online Hierarchical Job Scheduling on Grids with Admissible Allocation. Journal of Scheduling 13(5), 545–552 (2010)
21. Lifka, D.A.: The ANL/IBM SP Scheduling System. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1995 and JSSPP 1995. LNCS, vol. 949, pp. 295–303. Springer, Heidelberg (1995)
22. Skovira, J., Chan, W., Zhou, H., Lifka, D.: The EASY - LoadLeveler API Project. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1996 and JSSPP 1996. LNCS, vol. 1162, pp. 41–47. Springer, Heidelberg (1996)

Expression Templates and OpenCL

Uwe Bawidamann and Marco Nehmeier

Institute of Computer Science, University of Würzburg
Am Hubland, D 97074 Würzburg, Germany
nehmeier@informatik.uni-wuerzburg.de

Abstract. In this paper we discuss the interaction of expression templates with OpenCL devices. We show how the expression tree of expression templates can be used to generate problem specific OpenCL kernels. In a second approach we use expression templates to optimize the data transfer between the host and the device which leads to a measurable performance increase in a domain specific language approach. We tested the functionality, correctness and performance for both implementations in a case study for vector and matrix operations.

Keywords: GPGPU, OpenCL, C++, Expression templates, Domain specific language, Code generation.

1 Introduction

In the last years computer architecture has changed from a single-core design to a multi-core architecture providing several processor cores on a single CPU. This new way of processor design has the intention to circumvent the physical constraints of increasing the performance of a single-core CPU by using symmetric multi-core processors to parallelize the computation [3].

Additionally the GPU (Graphics Processing Unit) has recently come into focus for general purpose computing by the introduction of CUDA (Compute Unified Device Architecture) [14] as well as the open standard OpenCL (Open Computing Language) [7] to exploit the tremendous performance of highly parallel graphic devices.

CUDA is NVIDIA's parallel computing architecture on GPU's which could be used for GPGPU (General Purpose Computing on Graphics Processing Units) through CUDA C [13], CUDA Fortran [18] or OpenCL [16].

OpenCL is an open standard for general purpose parallel programming across CPU's, GPU's and other processors [7]. It provides a C like programming language to address the parallel concept of heterogeneous systems. In contrast to CUDA C, which is the common programming language for CUDA devices, OpenCL is not limited onto a specific GPU architecture. OpenCL is rather available for NVIDIA CUDA GPU's [16], multi-core CPU's and the latest GPU's from AMD [2], Intel Core CPU's [6], DSP's [7] and many other architectures.

Both technologies, CUDA as well as OpenCL, have a huge impact onto the world of scientific computing and also scientists without an access onto a super

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 71–80, 2012.

computer as well as the normal user could benefit from the tremendous performance of GPU's in their workstation and desktop computers.

Alongside the use of standard applications which support CUDA or OpenCL, users are able to write their own applications running on GPU's. But this is burdened with many new concepts and techniques which are necessary to address the parallel programming on GPU's or heterogeneous systems. The programmer has to take care about the data transfer between the host system and the CUDA or OpenCL device, he has to organize the thread local or shared memory as well as the global memory and he has to use specific parallel programming techniques [12,15].

All these new concepts and techniques could be a real challenge for unexperienced developers in the area of GPGPU. In the worst case the program on a GPU has a inferior performance compared to a plain old sequential approach on a CPU caused by the bottleneck of inappropriate data transfers between host and device, misaligned memory access and bank conflicts, inappropriate load balancing, deadlocks and many more.

Hence, our intention was to investigate techniques and concepts to provide user-friendly libraries with an interface completely integrated into C++ hiding all the GPGPU specific paradigms from the user. Additionally the C++ concept of operator overloading offers the possibility to integrate the library interface as a domain specific language (DSL). Furthermore expression templates [19] are used to optimize the data transfer between the host and the device.

In this paper we used OpenCL as the hidden layer under the C++ interface using operator overloading and expression templates to utilize the benefits of GPU's as well as of multi-core CPU's. As an application area we chose matrix and vector operations which are particularly suitable to show the capability of a domain specific language embedded into C++ using expression templates and OpenCL.

2 Expression Templates

One of the outstanding features of C++ is the feasibility to overload several operators like the arithmetic operators, the assignment operator, the subscript operator or the function call operator. This offers a developer of particularly mathematical data types to implement the interface of this types as a domain specific language embedded into C++. But besides the possibility to define an easy and intuitively usable data type, the operator overloading in C++ has a drawback called pairwise evaluation problem [21]. This means that operators in C++ are defined as unary or binary functions. For expressions with more than one operator like `r = a + b + c` this has the consequence that `a + b` are evaluated first and than their result, stored in a temporary variable, is used to perform the second operator to evaluate the addition with the variable `c`. Obviously this leads to at least one temporary variable for each `+` operator but if the data types are e.g. vectors it also performs several consecutive loops to perform the particular vector additions. Both the temporary variables as well

as the consecutive loops are a performance penalty compared to a hand coded function computing the result with a single loop and no temporaries [21], see Listing 1.

```
for (int i = 0; i < a.size(); ++i)
    r[i] = a[i] + b[i] + c[i];
```

Listing 1. Hand coded and optimal computation of `r = a + b + c`

Expression templates [19] are a C++ technique to avoid this unnecessary temporaries as well as the consecutive loops by generating an expression tree composed of nested template types. Furthermore this expression tree is explicitly visible at compile time and could be evaluated as a whole resulting in a similar computation as shown in Listing 1. This could be achieved by defining a template class `BExpr<typename T, class OP, typename A, typename B>` as an abstract representation of a binary operation[1] specified by the template parameter `OP` as an exchangeable policy class [1] working on the arguments of the types `A` and `B` which are stored as reference member variables.

```
template<typename T, class OP, typename A, typename B>
class BExpr {
    A const& a_;
    B const& b_;
  public:
    BExpr(A const& a, B const& b) : a_(a), b_(b) { }

    T operator[] (size_t i) const {
        return OP.eval(a_[i], b_[i]);
    }
    ...
};
```

Listing 2. Class `BExpr<typename T, class OP, typename A, typename B>`

Thereby the template type `T` specifies the type of the vector elements which is important to implement the element-wise evaluation using the subscript operator. In principle the subscript operator uses the policy class specified by the template parameter `OP` to compute the $\texttt{i}^{\text{th}}$ element of the result of the operation. Listing 3 shows an implementation of a policy class for a vector addition.

```
template<typename T> struct Add {
    static T eval(T a, T b) { return a + b; }
};
```

Listing 3. Policy class `ADD<typename T>`

[1] Types for operations with a different order are defined in a similar manner. Furthermore we have shown in [10] how a general type for operations of a arbitrary order could be specified using a C++0x.

With these building blocks it is quite easy to overload the arithmetic operators for vector and scalar types to implement the operations like the vector sum or the multiplication with a scalar returning an expression tree. Figure 1 shows the composed tree structure for the expression `Scalar * (Vector + Vector)`. Therefore expression templates are a technique to compose expression trees with nested template classes at compile time.

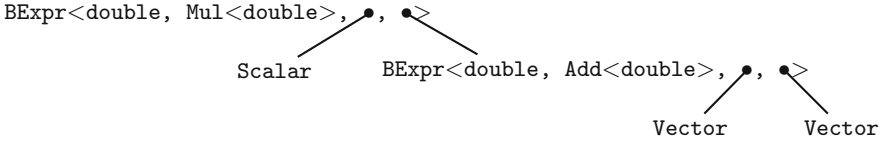


Fig. 1. Tree structure for the expression `Scalar * (Vector + Vector)`

The evaluation of such an expression tree then could be performed as a whole by overloading the assignment operator. For the example with vector operations it is quite easy by using one loop over the size of the vectors computing the result element-wise by calling the overloaded subscript operator of the tree.[2]

Obviously there are a lot of small objects and inline functions used for the expression template approach but due to modern compilers this evaluation leads almost to a similar machine code as Listing 1.

Using expression templates for vector operations to avoid unnecessary loops and temporaries is the classical example used in [19] to introduce these powerful technique. Besides loop fusion the expression templates are used in many other areas like the adaptation to grids in finite element methods [5] or to increase the accuracy in dot product expressions [8].

In addition we are working in the area of interval arithmetic and primarily used expression templates to reduce the necessary but time-consuming rounding mode switches of the floating point unit to speed up the computation [11]. Because this showed promising results we investigated an optimized expression template approach and combined it with additional functionality like the automatic differentiation [10] at compile time using template meta programming [20].

3 Implementation

Recently we started to investigate the usage of GPU's for reliable computing and especially interval arithmetic. Hence, the tremendous computing power of GPU's motivated us to research the potential of a GPU as a back end for a C++ interval library which is based on expression templates and template meta programming. Our goal in this context is to offer the user a domain specific language for interval computations which is able to utilize GPU's for specific

[2] Note that for this approach it is required that the `Vector` class as well as the `Scalar` class overload the subscript operator. For a `Scalar` it could easily return the value of the scalar.

algorithms or problem solvers. The aim of this work was to have a case study for the combination of expression templates and GPU's.

We chose OpenCL in place of CUDA as the framework for our case study because it offers the possibility to generate and compile the OpenCL code at run time and additionally it is possible to run it on different GPU's as well as on CPU's.

As a field of application we chose vector and matrix operations for the case study and we implemented two different approaches. The first one is to use expression templates to generate specific OpenCL code for the expressions. The second one uses precompiled OpenCL kernels which are utilized by the expression templates. Both approaches are compared against an implementation using standard operator overloading to realize the domain specific language in C++ and computing the result on the GPU.

3.1 OpenCL Code Generation

The approach of generating specific OpenCL kernels with expression templates needs to address three different problems. The transfer of the data between the host and the device, the unique mapping between the data and the kernel parameters and the generation of the kernel itself.

In a domain specific language approach there are two different possibilities to realize the transfer of the data onto the device. The first one is to allocate and fill the `cl::Buffer` on the device during the creation of the vector or matrix. For this, the two OpenCL memory flags `CL_MEM_READ_WRITE` and `CL_MEM_COPY_HOST_PTR` are used to initialize a read-write buffer with data from the host. The second approach is to allocate the buffer and copy the data within the assignment operator prior to the execution of the kernel. Both of them have their advantages and disadvantages. The first one has to copy the data only once but the required memory, which is limited on a GPU, is used all the time and the access of the data from the host is more costly. The second one has to copy the data for each expression but it only uses the required memory for the ongoing expression and the access from the host is easy.

For the unique mapping of the data and the kernel parameters we used the singleton pattern [4] to implement a class to generate unique id's which are requested at the construction of the vectors and matrices and stored as the member `id_`. Additionally the vector and matrix classes provide two methods `getIdent` and `getCode` which return the string ``v'' + id_` as an identifier and the string ``v'' + id_ + ``[index]'' as the code snippet to access the data. Both methods are required to generate the specific OpenCL kernels out of the expression tree inside the assignment operator and to access the required data. For scalar types it is not required to return the identifier. In this case the method `getCode` returns the value of the scalar itself which is included as a constant into the kernel code.

The generation of the kernel itself is then subdivided into three parts. The first one is the computation of all required parameters. For this task an instance

`paramSet` of the STL [17] container type `std::set` is recursively filled with all vectors or matrices of the expression.[3]

Then afterwards this `paramSet` could be used to generate the header of the kernel while iterating over the set declaring the parameters of the expression, see Listing 4. Additionally a constant for the index of the work item is declared.

```
std::string code = "__kernel void CLETGenVectorKernel( ";
for ( it = paramSet.begin() ; it != paramSet.end(); it++ ) {
    code += "__global float* " +
          (*it).getObject().getIdent() += ", ";
}
code += " const unsigned int size )" +
    "\n{ \n" +
    " const int index = get_global_id(0); \n";
```

Listing 4. Generation of the kernel header and the constants

The last but most important part is to generate the body of the kernel itself. This is done by calling the method `getCode` for the result type as well as for the expression tree, see Listing 5.

```
code += " " + result.getCode() + " = " +
    expression.getCode() + ";\n};\n \n";
```

Listing 5. Generation of the kernel body

Obviously the method `getCode` is a replacement of the subscript operator of the expression template implementation introduced in Section 2. Hence the nodes of the expression tree used for the code generation, which are almost similar to the class `BExpr` in Listing 2, have to implement the method `getCode`. But this is quite easy by using specific policy classes to concatenate the strings of the recursive call of the child nodes with the required arithmetic operator, see Listing 6.

```
std::string getCode() const {
    return std::string("(" + a_.getCode() +
        " + " + b_.getCode() + ")" );
 }
```

Listing 6. Method `getCode` of a tree node for the addition

Subsequently the generated kernel string could be compiled and executed using the OpenCL API functions inside the assignment operator.

3.2 Utilize Precompiled OpenCL Kernels

In addition to the implementation in Section 3.1 to generate specific kernels out of the expression tree we inspected the use of expression templates in place of

[3] The vectors or matrices are stored as a constant reference in a wrapper class to afford a sorted organization of the `paramSet`. The method `getObject` offers access to the stored object.

operator overloading to minimize the data transfer of a domain specific language using precompiled kernels on a GPU back end. The aim of this case study is to realize a DSL where some parts of the computation, e.g. vector or matrix operations, are executed on a GPU without the requirement to transfer the data between the host and the device manually by the user. A classical approach using operator overloading suffers from the pairwise evaluation problem, see Section 2, where data is transferred from the host to the device, then the result is computed on the GPU and transferred back onto the host for every single operation. With expression templates the unnecessary data transfers can be eliminated.

The implementation of the expression templates are almost similar to the approaches in Section 2 and Section 3.1 respectively. The most important difference is that in addition to the leaf nodes (vectors, matrices ...) of the expression tree all inner nodes of the tree are annotated with unique id's. These id's are required to map the several operations of the expression, performed by operation specific precompiled kernels, onto the temporaries used for the results of the operations.

The evaluation of the expression tree is as well performed inside the assignment operator. But in contrast to the implementation in Section 3.1 there is no OpenCL code generated. Rather two traversals of the expression tree are required.

The first traversal is used to allocate all the necessary memory for the leaf nodes (vectors, matrices ...) as well as for the temporary results of the operations of the inner nodes on the OpenCL device. For each leaf node with a new id a `cl::Buffer` is allocated and initialized from the host. Additionally the id of the leaf node as well as the associated `cl::Buffer` are stored in a instance `paramMap` of the STL container `std::map` [17]. This approach has the benefit that the data of a leaf node is transferred only once even if the same associated variable/parameter is used multiple times in an expression. For inner nodes of the tree the required memory is only allocated for the temporary results of the related kernel of the operation. These allocated memory areas are stored in the `paramMap` together with their id's.

The second traversal of the tree performs the computation of the expression on the OpenCL device by using the method `compute`[4] of the nodes recursively. This method could be subdivided into two parts. First of all the recursive calls of the method `compute` of the child nodes take place to evaluate the subexpressions. Afterwards the operation specific parts are performed, which could be also implemented with policy classes to generalize the code of the implementation, see Section 2. These parts are the determination of the required memory areas for the parameters[5] and the result of the operation using the `paramMap` and the id's of the nodes as well as the execution of the operation specific precompiled kernel.

[4] This is the correspondent method to the subscript operator or the method `getCode` of the implementations in Section 2 or Section 3.1, respectively.

[5] These are the memory areas which are initialized by the evaluation of the child nodes or which are defined by the leaf nodes during the first traversal.

After the second traversal the evaluation of the expression is finished and the result is stored in the associated memory area of the root node of the tree. These data is transferred back to the host and the allocated device memory is freed.

4 Experimental Results

We have tested our two implementations on a AMD Radeon HD 6950 GPU.

For the code generation approach (Section 3.1) it turned out that for standard vector operations the compile time for the generated kernel is out of scale for most of the problems. For example the time required for the expression `v1 + v2 + v1 * 5` is shown in Table 1. In this comparison the code generation approach as well as the operator overloading approach use device memory which is allocated and initialized during the construction of the vectors. Hence only the time for the compilation as well as for the execution of the kernel is measured. In contrast the operator overloading approach uses precompiled kernels.

Table 1. Performance comparison of the code generation approach (Section 3.1)

Vector size	Code generation	Operator overloading
4096	63 ms	1,2 ms
1048576	63,2 ms	1,3 ms
16777216	64,8 ms	5,7 ms

However, if we don't regard the compile time, we have measured an execution time of 3 ms for a vector size of 16777216 which is almost the half of the execution time with operator overloading. Hence, the code generation approach could be a good choice for hard problems where the compile time is almost irrelevant.

Table 2. Performance comparison of the precompiled kernel approach (Section 3.2)

Expression	Precompiled kernels	Operator overloading
`(M1 + M2) + (M3 + M4)`	26 ms	51 ms
`(M1 + M2) + (M3 + M1)`	21 ms	51 ms
`(M1 * M2) * (M3 * M4)`	183 ms	218 ms

On the other hand we have inspected the approach using precompiled kernels against an implementation using operator overloading for matrix operations. For this case study the data of both approaches are kept on the host and are only transferred for the execution of the expression template or for the particular operation, respectively. Table 2 shows the required time for the execution of three different expressions for a matrix size 2048×2048. Note that for the second expression where the matrix `M1` occurs two times the expression template approach is 5 ms faster than the first expression which also adds 4 matrices whereas the

operator overloading approach is identical. This speed up was reached because the matrix `M1` is in this approach transferred to the device memory only once.

For the third expression the speed up of the expression template approach is not such significant because the execution of the matrix multiplication on the GPU predominate the required latency time for the data transfer from the host to the device.

As a last remark to the execution time of OpenCL kernels we have compared the kernels on an AMD Radeon HD 6950 GPU against an Intel Core i7-950 CPU. Due to the latency time for the data transfer from the host to a GPU the execution of an expression is only worth for a big enough problem. For example the execution time including the data transfer for a matrix multiplication of two square matrices is almost similar for 256×256 matrices. For smaller matrices the CPU is significantly faster than the GPU. To avoid this runtime penalty we can use template meta programming[6] at compile time or normal branches at run time to decide, if we would run the kernels on the GPU or CPU, respectively.

5 Related Work

In [22] a analogical approach as in Section 3.1 is used to generate CUDA C for vector operations but they have only measured the pure execution time of the kernels neither with a regard for the time of the compilation nor the data transfer.

6 Conclusion and Further Research

In this paper we have shown that it is possible to use expression templates to build a bridge between a domain specific language in C++ and OpenCL. We have presented two different approaches.

The first one is to generate and compile the problem specific OpenCL kernels out of the expression trees. A drawback of this approach is the time required to compile the kernel, which is generated along with the evaluation of the expression template. On the other hand these specifically generated kernels showed a good performance for the pure execution time on the GPU. Hence, it could be a good choice for harder problems. Since expression templates are explicit types only one compilation of identical expression trees is required. Another application is to mix in the problem specific code into preassembled problem solvers, e.g. mix in the computation of a function and their derivative, using automatic differentiation, into a preassembled interval Newton method.

Our second implementation is used to reduce the necessary data transfers between the domain specific language and the OpenCL device by using precompiled kernels which are called in an optimized way. Thanks to the expression templates it has the benefit that the necessary data transfer is reduced to minimum.

[6] If the problem size is available at compile time.

Further investigations are planned to improve the interaction between domain specific languages, expression templates and template meta programming on the host and OpenCL and CUDA on the device to offer libraries for heterogeneous systems which are fast and easy to use.

References

1. Alexandrescu, A.: Modern C++ design: generic programming and design patterns applied. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
2. AMD: AMD accelerated parallel processing OpenCL programming guide, version 1.2c (April 2011)
3. Brodtkorb, A.R., Dyken, C., Hagen, T.R., Hjelmervik, J.M., Storaasli, O.O.: State-of-the-art in heterogeneous computing. Sci. Program. 18, 1–33 (2010)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
5. Härdtlein, J.: Moderne Expression Templates Programmierung. Ph.D. thesis, Universität Erlangen-Nürnberg (2007) (in German)
6. Intel: Intel OpenCL SDK user's guide, document number 323626-001US (2011)
7. Khronos OpenCL Working Group: The OpenCL Specification, version 1.1.44 (June 2011)
8. Lerch, M., Wolff v. Gudenberg, J.: Expression templates for dot product expressions. Reliable Computing 5(1), 69–80 (1999)
9. Lippman, S.B. (ed.): C++ Gems. SIGS Publications, Inc., New York (1996)
10. Nehmeier, M.: Interval arithmetic using expression templates, template meta programming and the upcoming C++ standard. Computing 94(2), 215–228 (2012), http://dx.doi.org/10.1007/s00607-011-0176-6
11. Nehmeier, M., Wolff von Gudenberg, J.: filib++, Expression Templates and the Coming Interval Standard. Reliable Computing 15(4), 312–320 (2011)
12. NVIDIA: NVIDIA CUDA C best practices guide, version 3.2 (August 2010)
13. NVIDIA: NVIDIA CUDA C programming guide, version 3.2 (November 2010)
14. NVIDIA: NVIDIA CUDA reference manual, version 3.2 Beta (August 2010)
15. NVIDIA: OpenCL Best Practices Guide (May 2010)
16. NVIDIA: OpenCL programming guide for the CUDA architecture, version 3.2 (August 2010)
17. SGI: Standard Template Library Programmer's Guide, April 20 (2011), http://www.sgi.com/tech/stl/
18. The Portland Group: CUDA Fortran programming guide and reference, version 11.0 (November 2010)
19. Veldhuizen, T.: Expression templates. C++ Report 7(5), 26–31 (June 1995), reprinted in [9]
20. Veldhuizen, T.: Using C++ template metaprograms. C++ Report 7(4), 36–43 (1995), reprinted in [9]
21. Veldhuizen, T.: Techniques for scientific C++. Tech. Rep. 542, Indiana University Computer Science, version 0.4 (August 2000)
22. Wiemann, P., Wenger, S., Magnor, M.: CUDA expression templates. In: Proceedings of WSCG Communication Papers 2011, pp. 185–192 (2011)

Portable Explicit Threading and Concurrent Programming for MPI Applications

Tobias Berka[1], Helge Hagenauer[1], and Marian Vajteršic[1,2]

[1] Department of Computer Sciences,
University of Salzburg, Salzburg, Austria
[2] Department of Informatics, Mathematical Institute,
Slovak Academy of Sciences, Bratislava, Slovakia
{tberka,hagenau,marian}@cosy.sbg.ac.at

Abstract. New applications for parallel computing in today's data centers, such as online analytical processing, data mining or information retrieval, require support for concurrency. Due to online query processing and multi-user operation, we need to concurrently maintain and analyze the data. While the Portable Operating System Interface (POSIX) defines a thread interface that is widely available, and while modern implementations of the Message Passing Interface (MPI) support threading, this combination is lacking in safety, security and reliability. The development of such parallel applications is therefore complex, difficult and error-prone. In response to this, we propose an additional layer of middleware for threaded MPI applications designed to simplify the development of concurrent parallel programs. We formulate a list of requirements and sketch a design rationale for such a library. Based on a prototype implementation, we evaluate the run-time overhead to estimate the overhead caused by the additional layer of indirection.

Keywords: Threads, Parallel programming, Concurrent programming, Message passing interface.

1 Introduction

Multi-core processors and fast Infiniband networks have become off-the-shelf commodity components in todays data centers. Consequently, there is now an enormous potential for parallelism in enterprise computing. Due to the high computational demands, online analytical processing (OLAP), data mining and information retrieval are ideal candidates for the use of parallel computing.

Traditionally, parallel programs were written primarily for conventional supercomputing applications, such as numerical simulations. But with the advent of multi-core CPUs in mainstream server infrastructure, the availability of parallel computer systems has opened up new vistas in the development of parallel software. There are now many new areas of interest including data mining, online analytical processing (OLAP), information retrieval (IR) and many other economically attractive application domains. However, these applications have

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 81–90, 2012.

very different requirements compared to conventional parallel numerical simulations and kernel functions. For one thing, we are no longer dealing with batch jobs, which are deployed, perform their computation, deliver their result and terminate. These new applications must operate for extended durations, and have therefore been referred to as *persistent parallel services*. This causes some immediate problems related to fault tolerance, because the traditional error recovery mechanisms are inadequate for such applications [1]. Another requirement that we believe is currently not sufficiently supported is the ability to increase the number of hosts without halting or suspending the parallel application. This type of dynamic growth is required to scale the system as the problem size increases.

But in this paper, we are concerned with another problems that arise in a number of online data analysis activities: the need for *concurrent processing*. As we migrate data mining, OLAP or information retrieval to a parallel setting, we must maintain the data and execute queries *concurrently*. The information we operate on may grow, shrink or change spontaneously. As such, we must ensure consistency for concurrently executed queries. In addition, if we allow multi-user operation, multiple queries may arrive simultaneously. We thus require means to concurrently receive, store and distribute processing requests. If we can execute these queries in parallel, we must prevent interference on the communication channels and isolate the individual workloads. In this paper, we propose an additional layer of portable middleware on top of the most commonly used middleware for parallel programs on distributed memory computer systems – the message passing interface (MPI). It should provide concurrency through threads in a safe, reliable and convenient manner.

2 The MPIT Programming Model

The central key concept of our API is the use of threads to isolate concurrent activities in parallel MPI programs. The MPIT API captures this intent in the **thread collective** pattern. A thread collective is a parallel ensemble of threads, which cooperate across MPI processes in a parallel activity, that has to be isolated in terms of execution and communication. Figure 1 depicts the basic deployment pattern.

It consists of one thread on every process of an MPI communicator. The collective receives a *private MPI communicator*, which is copied from the source communicator. The threads of the collective can communicate with each other on this isolated communication channel. Communication on higher-level communicators such as MPI_COMM_WORLD is possible, but not encouraged, to avoid collisions. For local inter-thread communication, the MPIT API provides a set of reliable thread synchronization mechanisms to support the development of correctly synchronized programs. The threads of a thread collective are intended to be used primarily for long running activities – the rapid creation and joining of sub-threads is not a primary goal.

In our programming model, every thread is characterized by two memberships: the process and the collective. For threads of identical or different process

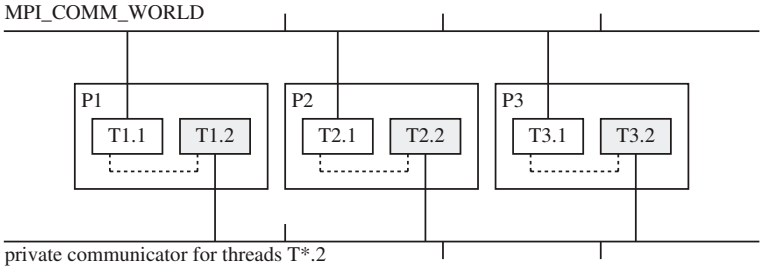


Fig. 1. Illustration of the MPI Threads (MPIT) programming model for three processes P1–P3 and two threads per process. MPI communicators are depicted as busses. Dotted lines indicate that threads can communicate via shared memory. Threads are used to isolate concurrent activities in parallel MPI applications. Consequently, the parallel thread collective consisting of T1.2, T2.2 and T3.2 has an isolated communication channel. Communication between local threads, e.g. T1.1 ↔ T1.2, takes place through shared memory and the shared address space, e.g. that of process P1.

and collective, we can discriminate four cases of thread communication. However, only two of these modes of communication are encouraged. Firstly, for communication amongst *local* threads from *different* thread collectives we can use the common address space of the common process. Secondly, communication with *remote* threads of the *same* thread collectives takes place on the private MPI communicator. Thirdly, communication with *remote* threads from *different* thread collectives can be achieved on a higher-level MPI communicator, given that it is adequately used. However, this type of communication is not encouraged by the MPIT library and is not supported directly. The main motivation is that use of higher-level, non-private communicators can easily lead to incorrect programs. As an example, consider what would happen if all threads shown in Figure 1 attempt to perform a broadcast on the global communicator MPI_COMM_WORLD. The result would be that six independent calls of MPI_Bcast would be made on an MPI communicator with a size of three. Consequently, the threads would randomly participate in one of two independent, sequential broadcast operations. The fourth type of communication would be local communication within the same thread collective, which is impossible by definition because a thread collective consists of one thread per participating process. Instead, the last case refers to global communication with all other threads regardless of process and collective. However, this is currently not supported by the MPIT API. It is possible to provide a layer of communication on top of MPI that provides a rank for every individual thread – including collective operations such as broadcasts and reductions – but we have thus far considered this out of scope for our library.

As an example, consider a parallel search engine, which allows multiple users to concurrently add, modify or delete documents in the index and execute queries. This is a typical case for a system that requires some form of concurrency support. A very natural way of dealing with this situation is to use threads

and queues that handle these two types of operations. Both maintenance and query operations are handled by a separate thread, which has a queue to hold incoming requests. Coarse-grained locks, which protect not individual documents but groups thereof, can be used to protect the index data against concurrent modification. If each of these threads have their own MPI communicators, they can communicate freely without having to guard against interference. Therefore, our notion of a thread collective comes as a natural fit for this type of situation.

3 Related Work

Threading is no new topic in the field of parallel programming. In literature, we have identified three main directions related to our work: use of threads in parallel programming paradigms, threads in MPI programs and use of POSIX threads in parallel programming.

The parallel programming community has developed a range of inherently multi-threaded programming models. A full survey of shared address space models and techniques is beyond the scope of this paper, but let us briefly mention two representative technologies. OpenMP is one of the most prominent examples [2] [3]. Originally developed primarily for shared memory systems, it has since been extended to distributed memory settings, see e.g. [4] [3]. Unified parallel C is a dialect of the C programming language which supports explicit parallel programming with a shared address space [5]. The original focus was on non-uniform, remote memory access for machines with a single thread per CPU, but has since been extended to multi-core machines [6]. But all of these programming environments are aimed at the field of parallel programming. For our purposes we require portable means to implement concurrent activities in a safe and effective manner. Parallel programming is of course the key feature of any persistent parallel service, but concurrency support is required on top of a parallel environment. Therefore, multi-threaded parallel programming models can only be seen as complementary base technologies – and not as rivals.

MPI is an interface specification for the message passing parallel programming model. It is often seen as the canonical representative of middleware for parallel processing on distributed memory systems. Traditionally, MPI's primary unit of organization is the process. By definition, a process is an abstraction that implies a private address space. If shared memory is available on a particular hardware platform, notably multi-core processes, an MPI implementation may use it to provide effective inter-process communication for processes located on the cores of an individual CPU. While this is effective enough to compete head-on with OpenMP and UPC [7], the sole purpose of this optimization is to make efficient use of contemporary CPU designs. The primary unit of processing is and remains that of a process. However, recognizing the need to support concurrency in the form of threads, earlier versions of the MPI standard have been augmented to permit processes to consist of multiple thread. In its current version, the key issue for thread support is the thread safety of all MPI functions. Many existing MPI implementations have reached full thread support with a thread support

level of MPI_THREAD_MULTIPLE, but the development of high-performance MPI implementation with full thread support continues to be a research issue [8] [9]. Based on this thread support, MPI has been combined with existing multi-threaded parallel programming models. The combination of OpenMP and MPI is still an ongoing research issue [10] but not a new idea [11]. But these are hybrid programming models and do not provide us with the concurrency support we require. The MPI Forum currently leads an open discussion of new features for MPI 3.0. For performance reasons, collective operations are to be supported asynchronously [12]. This requires the use of threads on all nodes to carry out the work required by the collective operation, while the user program continues with its processing. All in all, we can say that MPI has developed support for threads to the point that our necessary preconditions have been met. But the available facilities are by no means sufficient for our needs. We thus need to consider other means for acquiring the required functionality.

When we think of a threading environment for parallel software, the POSIX thread environment immediately comes to mind [13]. This thread interface has seen much use for the development of thread-based parallel programming tools, such as unified parallel C [14]. But for a number of reasons, we do not believe that this is sufficient.

For one thing, threads are not supported by the operating system on all high-performance platforms, but user-level threads can be used to provide the required functionality on such systems [15]. The Cray XT4 system [16] has been cited as an example for a thread-less modern system. Unfortunately, the lack of threads causes problem in modern extensions to MPI [12]. By defining an independent interface we can use a wider range of implementation techniques for threading without having to modify the applications. In another matter, the POSIX threads API defines unreliable semantics for condition variables. When waiting on a condition variable, a thread may experience *spurious wake-up*, meaning that the thread can be awakened without any external signal or other stimulus. This decision is justified by gains in performance [13]. However, due to the general discussion surrounding the difficulty of both parallel programming [17] and concurrent programming [18], we believe that unreliable semantics in the thread synchronization system are not going to aid the advancement of the state of the art. A more subtle issue is that the use of the POSIX thread interface in an MPI program forces us to mix two very different styles of programming. MPI defines and adheres to certain coding convention, which have been introduced to impose a clear structure on MPI programs. The POSIX thread interface, on the other hand, adheres to the conventions of UNIX system programming. The differences are most obvious in the error handling code. Any MPI function call will return MPI_SUCCESS if it completes normally, and we normally do not expect to require much in the way of error handling once a parallel application has reached a mature state. POSIX thread function calls have very different error handling requirements, and we are sometimes forced to analyze the error codes returned by a function call in order to ensure a deterministic behavior, e.g. due

to spurious wake-up on condition variables. Therefore, these differences are more than just a *cosmetic* issue, and we believe that it is preferable to have a thread interface that conforms to the same coding style and conventions as MPI.

4 The MPI Threads API

The analysis of the related work indicates that while the prerequisites for concurrency by threading in MPI programs have been met, no existing technique is a natural fit for the problems we face in implementing persistent parallel services. But the shortcomings of these methods provide a set of requirements for our approach.

We require a set of explicit thread control primitives instead of a thread-based parallel programming model. Since we need to integrate our solutions with existing MPI programs, we require a library that is fully compatible with the C programming language. The concurrent programming community argues that there are many good reasons to prefer language mechanisms over APIs [19], notably the ability to enforce correctness and to avoid certain performance penalties, but practical necessities suggest otherwise. Such an API should be designed to be similar in style to the MPI interface definitions, especially in terms ofreliability and error handling. Due to issues related to determinism, reliability and coding conventions, we consider the POSIX thread interface insufficient for our purposes. The key goal of such a development should be the interface specification and not the implementation, because it may be necessary to implement it for platforms without any out-of-the-box thread support. In addition, the interface declaration and the implementation should be decoupled to the largest extent possible, so that implementors can re-use existing definitions. Users of our library should be able to combine any two implementations of our thread middleware and the MPI interface standard. This means the we must be able to work with all MPI implementations that provide a thread level of `MPI_THREAD_MULTIPLE` and avoid any implementation-dependent features. With these goals in mind, we have developed the "MPI Threads API" or MPIT API – an interface definition and a prototype implementation in the C programming language. It provides the following features:

- *Thread* control – the ability to create threads, temporarily suspend them and to either join them with or detach them from their parent thread.
- Mutual exclusion or *mutex* locks – which can be atomically acquired by only one single thread at a time and remain in their possession until they are released.
- *Condition variables* – objects on which threads can be suspended from their execution until they are explicitly reactivated by another thread.
- *Barriers* – synchronization objects that block arriving threads until all participating threads have arrived.

The prototype internally uses the POSIX threads interface. On a modern POSIX compliant operating system it only depends on an MPI implementation. Since

MPIT is based on POSIX threads, it is fair to ask what benefits one can expect to gain from this additional layer of indirection. In its current version, the MPIT library provides the following advantages:

- Re-use of objects that have previously been released is detected.
- All return codes are consistently being checked for all possible errors including depletion of memory and operating system resources.
- The API conforms with MPI coding conventions.
- Thread cancellation as provided by the POSIX specification has not been included and is restricted to explicit shut-down.
- Collective thread creation across all processes on an MPI communicator is supported to aid the construction of concurrent parallel programs.
- Full error checking for mutex objects is enabled by default.
- Condition variables are no longer subject to spurious wake-up.
- Barriers are provided as part of the regular synchronization functionality.
- The participating threads of a barrier are explicitly identified, wheres POSIX threads merely count the participants without considering their identity.
- The MPIT library provides a *dynamic barrier*, where threads may safely join or leave the set of participating threads at their discretion, as we will discuss below.

Summarizing, we can say that the MPIT API extends or modifies the semantics of conventional POSIX threads in a number of ways. The typical approach for most systems will be to implement it on top of the POSIX thread layer. Despite all the benefits it offers, the our library constitutes a layer of indirection which cannot be as efficient as a native implementation. Let us therefore examine the overhead in execution time.

5 Evaluation

Our prototype implementation of the MPIT interface has been built as a user space library using the native POSIX thread control primitives. It wraps POSIX system calls and adds the functionality outlined above, and thus constitutes an additional layer of indirection. This indirection necessarily comes at the cost of an increased execution time. To quantify and report this cost, we have conducted a performance evaluation for three use cases. For reasons of scope we cannot present the native POSIX threads implementation in this paper, but it is derived in a straightforward manner from the following three MPIT-based program.

In the first experiment, we have measured the time to create, execute, terminate, join and free an empty thread function that consists only of a return statement. The second experiment extends the first by adding a mutex and having all spawned threads lock and unlock it repeatedly. This experiment is designed to measure the execution time of mutex operations. The third experiment measures the time it takes to perform wait or wake operations on a condition variable. Unfortunately, we could not compare the performance of our barrier implementation to the POSIX thread barrier, because it is part of the advanced

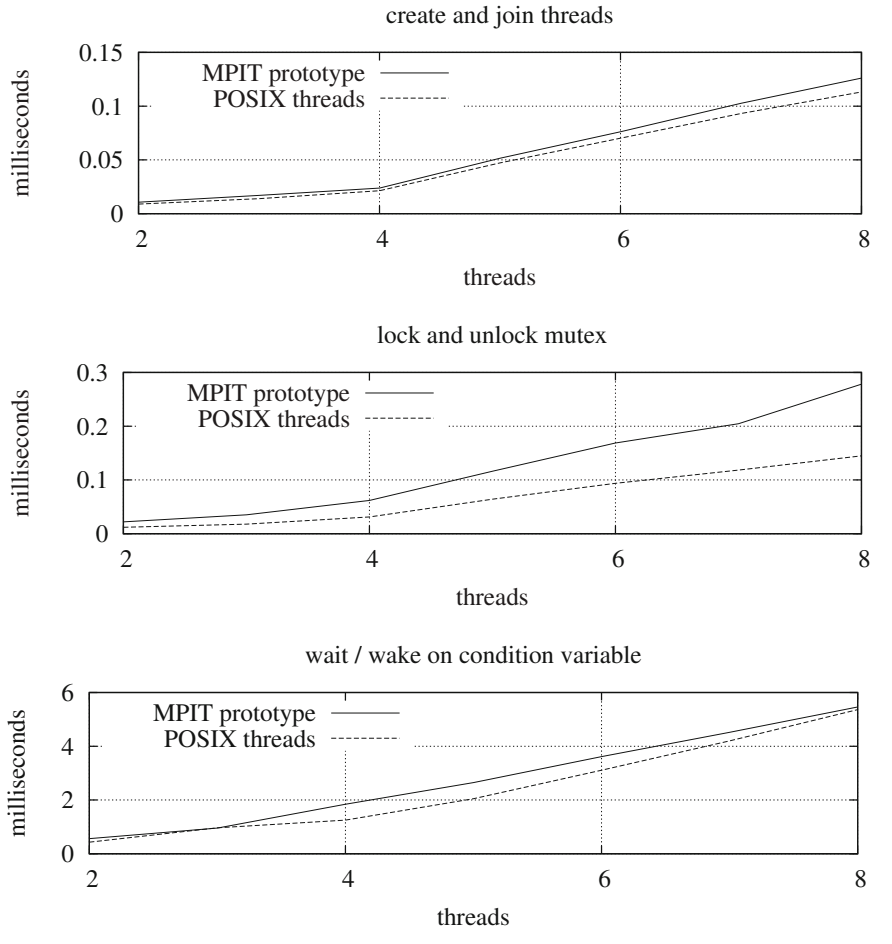


Fig. 2. Performance comparison between POSIX threads and the MPIT prototype implementation

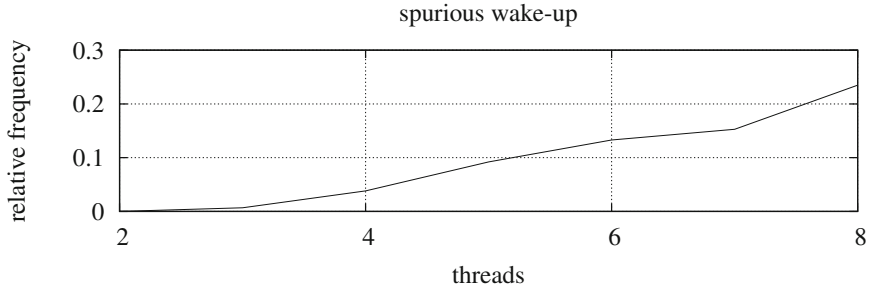


Fig. 3. Frequency of spurious wake-up events relative to all wait operations

real-time threads functionality and was not available on our evaluation platform. We have executed all of these tests on an system equipped with an Intel i7-920 processor running at 2.67 GHz with 6 GiB of main memory, CentOS 64 bit version 5.2, Linux kernel version 2.6.18-194.3.1.el5, and compiled and executed with MPICH2 version 1.0.6 in conjunction with the GNU Compiler Collection version 4.1.2.

The results for all three experiments are depicted in Figure 2. We have repeated all operations 1000 times, performed 100 independent measurements and found no signifiant deviations. These results indicate that our thread creation and condition variable operations do not suffer a substantial loss of performance compare to a direct implementation with POSIX threads. While the mutex operations suffer a greater slow-down, they remain very fast operations. To gauge the necessity of a reliable threading middleware we have conducted measurements to determine the relative frequency of spurious wake-up relative to the total number of thread reactivation events. Depicted in Figure 3, we can see that spurious wake-up is highly dependent on the number of threads using a condition variable. Based on these figures, the use of mechanisms to avoid spurious wake-up from occurring must be considered critical for highly contended systems.

6 Summary and Conclusion

In this paper, we have argued the need for an additional layer of middleware for parallel programs to support threading and concurrent programming for MPI-based parallel applications. We have outlined our requirements for such a middleware, which include explicit thread control instead of a thread-based parallel programming paradigm, a library-based instead of a language-based approach, conformance with MPI in terms of naming, error codes and other programming conventions, more reliable and deterministic behavior than the POSIX specification for inter-thread condition variables and inter-process semaphores and to produce a portable specification that can be used with any particular MPI implementation, and can be implemented on a wide range of thread systems including user-space threads. We have then given a full presentation of the MPI Threads API, which attempts to meet these requirements and constitutes the main contribution of this paper. Our future research objectives are three-fold. Firstly, we will seek to improve the efficiency of our current prototype implementation, which focuses on correctness, e.g. by using macros to provide a light-weight implementation of mutex objects. Secondly, we will investigate how our API can be used to implement a range applications requiring concurrent programming, e.g. queue-based workload distribution systems. Thirdly, we intend to apply the MPI Threads API to the development and deployment of persistent parallel services in information retrieval and data mining, to demonstrate the ability to construct viable systems in these application domains.

Acknowledgements. The third author was supported by VEGA grant no. 2/0003/11 from the Scientific Grant Agency of the Ministry of Education and the Slovak Academy of Sciences.

References

1. Latham, R., Ross, R., Thakur, R.: Can MPI Be Used for Persistent Parallel Services? In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 275–284. Springer, Heidelberg (2006)
2. Dagum, L., Menon, R.: OpenMP: An Industry Standard API for Shared-Memory Programming. IEEE Computational Science and Engineering 5(1), 46–55 (1998)
3. OpenMP Architecture Review Board: The OpenMP API Specification for Parallel Programming (2008)
4. Duran, A., Gonzàlez, M., Corbalán, J.: Automatic Thread Distribution for Nested Parallelism in OpenMP. In: Proc. ICS, USA, pp. 121–130 (2005)
5. UPC Consortium: UPC Language Specifications, v1.2. Technical Report LBNL-59208, Lawrence Berkeley National Lab (2005)
6. Zheng, Y., Blagojevic, F., Bonachea, D., Hargrove, P.H., Hofmeyr, S., Iancu, C., Min, S.J., Yelick, K.: Getting Multicore Performance with UPC. In: Proc. PP, USA. SIAM (2010)
7. Mallón, D.A., Taboada, G.L., Teijeiro, C., Touriño, J., Fraguela, B.B., Gómez, A., Doallo, R., Mouriño, J.C.: Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) PVM/MPI. LNCS, vol. 5759, pp. 174–184. Springer, Heidelberg (2009)
8. Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Thakur, R.: Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming. Int. J. High. Perform. C. 24, 49–57 (2010)
9. Thakur, R., Gropp, W.: Test Suite for Evaluating Performance of Multithreaded MPI Communication. Parallel Computing 35, 608–617 (2009)
10. Lusk, E., Chan, A.: Early Experiments with the OpenMP/MPI Hybrid Programming Model. In: Eigenmann, R., de Supinski, B.R. (eds.) IWOMP 2008. LNCS, vol. 5004, pp. 36–47. Springer, Heidelberg (2008)
11. Smith, L., Bull, M.: Development of Mixed Mode MPI / OpenMP Applications. Scientific Programming 9, 83–98 (2001)
12. Hoefler, T., Kambadur, P., Graham, R.L., Shipman, G., Lumsdaine, A.: A Case for Standard Non-Blocking Collective Operations, Germany. Springer (October 2007)
13. The Austin Group, The Open Group, The IEEE: ISO/IEC/IEEE 9945:2009 Information Technology – Portable Operating System Interface (POSIX©) Base Specifications (7) (2009)
14. Zhang, Z., Savant, J., Seidel, S.: A UPC Runtime System Based on MPI and POSIX Threads. In: Proc. PDP, USA, pp. 195–202. IEEE (2006)
15. Rodrigues, E.R., Navaux, P.O.A., Panetta, J., Mendes, C.L.: A New Technique for Data Privatization in User-Level Threads and its Use in Parallel Applications. In: Proc. SAC, USA, pp. 2149–2154. ACM (2010)
16. Alam, S.R., Kuehn, J.A., Barrett, R.F., Larkin, J.M., Fahey, M.R., Sankaran, R., Worley, P.H.: Cray XT4: An Early Evaluation for Petascale Scientific Simulation. In: Proc. SC, USA, pp. 39:1–39:12. ACM (2007)
17. Dongarra, J., Beckman, P., Aerts, P., Cappello, F., Lippert, T., Matsuoka, S., Messina, P., Moore, T., Stevens, R., Trefethen, A., Valero, M.: The International Exascale Software Project: A Call To Cooperative Action By the Global High-Performance Community. Int. J. High. Perform. C. 23, 309–322 (2009)
18. Buhr, P.A., Harji, A.S.: Concurrent Urban Legends: Research Articles. CCPE 17, 1133–1172 (2005)
19. Boehm, H.J.: Threads Cannot Be Implemented as a Library. SIGPLAN Notices 40, 261–268 (2005)

Verification of a Heat Diffusion Simulation Written with Orléans Skeleton Library

Noman Javed and Frédéric Loulergue

LIFO, University of Orléans, France
Firstname.Lastname@univ-orleans.fr

Abstract. Orléans Skeleton Library (OSL) provides a set of algorithmic skeletons as a C++ library on top of MPI. The parallel programming approach of OSL is a structured one, which eases the reasoning about the performance and correctness of the programs. In this paper we present the verification of a heat diffusion simulation program, written in OSL, using the Coq proof assistant. The specification used in proving the correctness is a discretisation of the heat equation.

Keywords: Parallel programming, algorithmic skeletons, verification of programs, proof assistant.

1 Introduction

With parallel machines being now the norm, parallel programming has to move to structured paradigms, in the same way sequential programming did more than forty years ago [12,17]. Algorithmic skeletons [10] and bulk synchronous parallelism [27] are two structured approaches of parallelism. In the former, a set of higher-order functions are provided to the user that combines them to build her parallel application, the skeletons capturing usual patterns of parallel programming. The latter imposes some restrictions on the form of parallel programs that make the reasoning about their performance simple yet realistic. The Orléans Skeleton Library or OSL [20], is a C++ library of bulk synchronous parallel algorithmic skeletons on top of MPI. It uses expression template techniques to ensure good performances.

Parallel machines being widespread, the verification of parallel programs becomes more and more important. In the case of parallel scientific applications, the guarantee of correctness of a program makes the users of rare or expensive computing resources, confident that these resources will not be wasted running programs with errors. We are thus interested on formalising and proving the correctness of OSL programs using Coq [3]. The Coq proof assistant provides a formal language to write mathematical definitions, executable algorithms and theorems. Its environment allows automated and interactive development of machine-checked proofs.

In this paper we study an application of heat diffusion simulation in one dimension. We will first give an overview of parallel programming with OSL

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 91–100, 2012.

(section 2) and discuss some related work (section 3), before presenting the application (section 4) and its formalisation and proof of correctness (section 5). We conclude and give future research directions in section 6.

2 Parallel Programming with OSL: An Overview

For the sake of conciseness we will not present the BSP model, nor the performance model of OSL skeletons. In the BSP model, any parallel computer is seen as a distributed memory machine with a fully connected network and a global synchronisation unit. The BSP machine is characterised by several parameters, including p the number of memory-processors pairs. OSL gives access to these parameters, including `osl::bsp_p`.

OSL skeletons manipulate *distributed arrays*. This data structure is a generic one: distributed arrays have type `DArray<T>`. One can seen a distributed array as an array containing elements of type `T`. In the implementation it is in fact a set of p *local* arrays, each one being hold by a processor. There are several constructors for building distributed arrays: creating a distributed array evenly distributed of a given size, and whose initial values are given by a constant or by a function from array indices to values ; creating a distributed array from a sequential array on the root processor (the local array on other processors being empty), *etc.* We will write informally, $[\ v_0\ ,\ldots,\ v_{n-1}\ ]$ for a distributed array of size n containing value v_i at index i.

It is not possible to access directly an element of a distributed array: we can only apply skeletons to it. We will not describe the whole OSL library here, but only the skeletons used in the heat diffusion simulation presented in the next section.

The `map` skeleton takes as argument a unary function f (strong typing is enforced as it is required that f extends `std::unary_function<A,B>`) and a distributed array $[\ v_0\ ,\ldots,\ v_{n-1}\ ]$ of type `DArray<A>`, and returns a distributed array of type `DArray<B>` that has the same distribution that the initial array and whose content is given by the following equation:

$$\texttt{map}\ f\ [\ v_0\ ,\ldots,\ v_{n-1}\ ] = [\ f(v_0)\ ,\ldots,\ f(v_{n-1})\ ]$$

The `zip` skeleton is similar to the `map` skeleton but it takes as argument a binary function f and two distributed arrays (that should have the same size and be distributed in the same way), and returns a distributed array that has the same distribution that the initial arrays and whose content is given by the following equation:

$$\texttt{zip}\ f\ [\ u_0\ ,\ldots,\ u_{n-1}\ ]\ [\ v_0\ ,\ldots,\ v_{n-1}\ ] = [\ f(u_0, v_0)\ ,\ldots,\ f(u_{n-1}, v_{n-1})\ ]$$

The `shift` skeleton is used for communications. It takes as argument an offset d, a replacement function f, and a distributed array $[\ n_0\ ,\ldots,\ n_{v-1}\ ]$. It returns a distributed array that has the same type and distribution that the initial array and whose content is given by the following equations:

$$\begin{array}{ll}\texttt{shift}\ d\ f\ [\ v_0\ ,\ldots,\ v_{n-1}\ ] = [f(0);\ldots;f(d-1);v_0;\ldots;v_{n-d-1}] & \text{if } d > 0\\ \texttt{shift}\ d\ f\ [\ v_0\ ,\ldots,\ v_{n-1}\ ] = [v_d;\ldots;v_{n-1};f(n-d-1);\ldots;f(n-1)] & \text{if } d < 0\end{array}$$

3 Related Work

There are many proposals for skeletal parallelism. A recent survey is [16]. Muesli and SkeTo [9,25] share with OSL a number of classical data-parallel skeletons on distributed arrays.

There is work on formal semantics of algorithmic skeleton languages or libraries: [11] is a data-parallel calculus of multi-dimensional arrays, but no implementation was released ; [1] is a formal semantics for the Lithium algorithmic skeleton language and it differs from OSL as it is a stream-based language ; the Calcium/Skandium library for Java has both a formal static and dynamic semantics [8], further extended in a shared memory context to handle exceptions [23] ; Quaff [14,13] is a highly optimised library with meta-programming techniques, and to attain its very good performances, some constraints are put on the algorithmic structure of the skeletons (that are mainly task parallel skeletons) ; Eden [18] is a parallel functional language often used to implement algorithmic skeletons [2]. To our knowledge, none of these semantics have been used as the basis of programs proofs of correctness, but are rather formal reference manuals for the libraries. Moreover, none have been formalised using a proof assistant. Using Coq, we provide an executable formal semantics (that allows testing with respect to the implementation) and a framework to prove the correctness of OSL programs, these proofs being machine-checked.

Program calculation [4,5,19] is the usual way to prove the correctness of algorithm skeletons programs (by construction), the proof being done "by hand". However a new algorithmic skeleton called BH as been implemented in Bulk Synchronous Parallel ML [24] and its implementation proved correct using the Coq proof assistant. This BH skeleton is used in a framework for deriving programs written in BH from specifications [15], and to *extract* BSML programs that could be compiled and run on parallel machines. A heat diffusion simulation directly written in BSML (therefore not at all in the algorithmic skeleton style) has also been proved correct and the BSML program extracted from the proofs [26].

4 A One Dimensional Heat Diffusion Simulation in OSL

The goal of the application is to give an approximate solution of the following partial differential equation describing the diffusion of heat in a one dimensional bar of metal of length 1:

$$\frac{\delta h}{\delta t} - \kappa \frac{\delta^2 h}{\delta^2 x} = 0 \qquad \forall t,\ h(0,t) = l \qquad \forall t,\ h(1,t) = r \tag{1}$$

where κ is the heat diffusivity of the metal, and l and r some constants (the temperature outside the metal). A discretised version of this equation is:

$$h(x,t+dt) = \frac{\kappa dt}{dx^2} \times \big(h(x+dx,t) + h(x-dx,t) - 2 \times h(x,t) \big) + h(x,t) \quad (2)$$

for a space step dx and a time step dt and with the same boundary conditions.

In sequential the temperature at the discretisation points in the metal bar is stored in an array, and a loop is used to update this array when a new time step occurs. For the OSL version, we use a distributed array for storing the values of h. If we consider the equation (2) for the whole array h, and not element-wise, then the update of h is is a linear combination of the distributed arrays in figure 1.

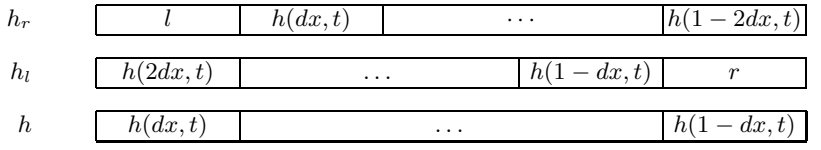


Fig. 1. Distributed Arrays for Heat Diffusion Simulation

Array h_r (resp. h_l) can be computed from array h using the `shift` skeleton with offset 1 (resp. -1) and with replacement function the constant function returning always l (resp. r). The linear combination of these arrays can be computed using the `map` and `zip` skeletons.

The OSL program for one step of update is thus:

```
bar =
  zip(std::plus<double>(),
      bar,
      map(boost::bind(multiplies<double>(),(kappa*dt)/(dx*dx),_1),
          zip(plus<double>(),
              zip(plus<double>(),
                  shift(1, leftBound, bar),
                  shift(-1, rightBound, bar)),
              map(boost::bind(multiplies<double>(),-2,_1), bar)))
```

where:

- `plus<double>()` is the addition on double precision floating point numbers of the C++ standard library,
- `boost::bind(multiplies<double>(),(kappa*dt)/(dx*dx),_1)` is a function obtained as partial application of the multiplication to the expression $\frac{\kappa dt}{dx^2}$,
- `leftBound` and `rightBound` are the constant replacement functions that return respectively l and r,
- and `boost::bind(multiplies<double>(),-2,_1)` is the function that multiplies by -2.

5 Verification Using the Coq Proof Assistant

For proving the correctness of the OSL heat diffusion simulation, we use the Coq proof assistant [3]. The Gallina language of Coq can be seen as a functional programming language with very rich types that allow to express logical properties about the programs. It is based on the Curry-Howard correspondence: types correspond to theorems and programs correspond to proofs. However, even if it is possible to write proofs as function programs, it is usually complex to do so. Therefore there also exists a tactics language used to interactively build proof terms. In this paper we will not present Coq source code (although the reader can access the complete code at `http://traclifo.univ-orleans.fr/OSL`), but we will sketch the formalisation and the proof in an informal manner.

5.1 OSL Programming Semantics: An Overview

The formalisation of the *programming model* of OSL in Coq is presented in [21]. This is a big-step semantics: it takes an OSL expression as input and returns in *one step* the value produced by this program. Small-step semantics may return several intermediate steps before reaching the final value. The OSL big-step semantics is formalised in Coq as a function evaluation that take as input a term representing an OSL program and returns either the value produced by this program or an error message.

The terms representing an OSL program are given by the following grammar, restricted here mostly to the skeletons used in the heat diffusion simulation, and without type information (in the Coq formalisation it is not possible to write OSL program wrongly typed ; it is therefore unnecessary to formalise a type system in addition to the syntax and proof that the typing works well with the operational semantics, it is the case by construction):

$$
\begin{array}{rcl}
e & ::= & se \;\mid\; pe \\
se & ::= & v \;\mid\; se\ se \;\mid\; \mathsf{reduce}(se, se, pe) \\
pe & ::= & pv \;\mid\; \mathsf{map}(se, pe) \;\mid\; \mathsf{zip}(se, pe, pe) \;\mid\; \mathsf{shift}(se, se, pe)
\end{array}
$$

An expression e could be either an expression se whose type is a sequential one, or a parallel expression pe. A sequential expression could be either a sequential value v (possibly a function), an application of a sequential expression to another sequential expression (for example to be able to apply a sequential function to the result of a parallel skeleton expression ended by a call to reduce), or an application of the reduce skeleton that takes as input a binary associative operator, an identity value for this operator, and a distributed array to be reduced. A parallel expression could be either a parallel value (the formalisation of a distributed array), a call to the map, zip or shift skeletons.

Values produced by OSL programs could be either sequential values (in case of the evaluation of the reduce skeleton), or parallel values, or an error message.

For the sequential part, we do not model the programming language: rather we consider than the sequential parts are black boxes (the v case of se), and are

modelled as Coq functions. Thus sequential types and values are formalised as the types and values of the functional programming language part of Coq. The syntax is polymorphic, and thus the user of this formalisation should choose the Coq type and values than corresponds the best to its C++ counterparts. For example one could choose to represent values of type int in C++ by values of type Z in Coq.

The parallel values are a formalisation of distributed arrays. This formalisation is a kind of structure/record that has three parts:

```
{
  content: list T;
  distribution: vector nat;
  invariant: length content = sum distribution
}
```

- the *content* of the distributed array, modelled as a Coq list of values of some type T
- the *distribution* of the distributed array, represented by the p sizes of the local arrays, therefore we use a data structure similar to a vector (of size p) of naturals for formalising distributions,
- a proof than the two first components are coherent: the length of the list representing the content of the distributed array should always be equal to the sum of elements of the vector representing the distribution of the distributed array. In Coq an expression of *type* length content = sum distribution is a *proof* of this equality.

5.2 Formalisation of the OSL Heat Diffusion Program

As we do not rely on the properties of the arithmetic operators, we only assume that we have a type number of numbers with usually operations $+$, $-$, $*$, $/$. When we write (op) for an operator, we consider it has a binary function that can be partially applied.

The formalisation of the OSL heat diffusion program, is therefore the expression shown in figure 2, provided bar is a distributed array of numbers, kappa, dt and dx are numbers, and leftBound and rightBound are constant functions from array index to numbers.

5.3 Proof of Correctness

The first step for the proof of correctness in the Coq proof assistant is to write the specification of the function that computes a step of the heat diffusion simulation, as indicated by equation (2) but with formally taking into account the boundary conditions. We model the array of temperatures by a list of numbers. For this list structure we have a function nth that takes as input an index i, a list, and a default value d: it returns the value in the list at index i, if the index is valid, and it returns the default value otherwise. It is a kind of array access but with

```
zip( ( + ),
     bar,
     map ( ( ∗ ) ( / ) ( ( ∗ ) kappa dt ) ( ( ∗ ) dx dx ),
           zip ( ( − ),
                 zip ( ( + ), shift(1, leftBound, bar), shift(−1, rightBound, bar) ),
                 zip ( ( + ), bar, bar ) ) ) )
```

Fig. 2. Modelling of OSL Heat Diffusion Program

the default value in case the index is not valid. The specification could then be written, for non-empty lists *bar* and valid indices i (these conditions are omitted here but not in the formal development):

$$\begin{array}{l}\forall \kappa\ dx\ dt\ l\ r\ bar\ i\ d,\\ \quad \mathsf{nth}\ i\ (\textbf{step}\ \kappa\ dx\ dt\ l\ r\ bar)\ d =\\ \quad \kappa \times dt/(dx \times dx) * \big(\\ \qquad (\mathsf{nth}\ (1+i)\ bar\ r) +\\ \qquad (\textbf{if}\ i = 0\ \textbf{then}\ l\ \textbf{else}\ \mathsf{nth}\ (i-1)\ bar\ d) -\\ \qquad ((\mathsf{nth}\ i\ bar\ d) + (\mathsf{nth}\ i\ bar\ d))\big) +\\ \quad (\mathsf{nth}\ i\ bar\ d).\end{array}$$

The function step takes as argument the numbers κ, dt, dx, l and r as well as a list of numbers *bar* and returns an updated list of numbers. For the parallel version of the heat diffusion simulation, we can use the expression of figure 2 and the evaluation function that models the OSL programming model as a big-step semantics. However what we can obtain directly from these two components is a function that takes as arguments the same numbers but a *distributed array* instead of a list and a returns either a distributed array or *an error message*.

The first problem can be easily solved: we compose the obtained function with a distributedArrayOfList function that takes a list and returns an evenly distributed array.

The second problem needs a *proof* that the evaluation of the expression of figure 2 will not raise an error. As a matter of fact, there is only one skeleton in this expression that can lead to an error message: the zip skeleton in case its two distributed array arguments do not have the same distribution. We have a short lemma that states that an evaluation of zip does not raise an error message when the two distributed arrays have the same distribution:

```
Lemma zipGivesOkForSameDistribution:
  ∀(A B C:Type)(f: A→B→C)
    (da1: distributedArray A)(da2: distributedArray B),
    distributedArray_distribution da1 = distributedArray_distribution da2 →
    exists d: distributedArray C, (zip f da1 da2 = Ok d ∧
      distributedArray_distribution d = distributedArray_distribution da1).
```

This results is used together with the fact that the other skeleton preserve distribution to prove that the evaluation of the OSL heat diffusion program does

not raise an error. From this proof we can build a function that evaluates the expression of figure 2 and returns a distributed array.

The main theorem thus states that this last function follows the specification defined above. The proof of this theorem proceeds as follows:

- the skeletons preserve distribution, so the guard of the application of zip can be removed,
- we then obtain for the content part of the result an expression similar to figure 2 but on lists instead of distributed arrays: using results on the commutation of nth with other functions on lists such as map, this expression can be simplified in such a way that the call to nth on the left hand side of the equation is not on top, but rather copied inside the expression,
- finally with reason by cases on i: i is 0, i is the length of *bar* minus 1, or i is in the middle. For each case the assumptions are enough to simplify both side of the equation to the same expression.

6 Conclusion and Future Work

High-level parallel programming approaches such as algorithmic skeletons ease the development of parallel programs. As parallelism is now everywhere, it is also important to be able to prove the correctness of parallel programs. Algorithmic skeleton programs are also easier to prove correct than programs written using Pthreads and MPI libraries. Based on a formal programming model of the Orléans Skeleton Library, we prove the correctness of one step of a heat diffusion simulation with respect to the *discretised* heat equation as specification.

If we consider the partial differential equation as the specification, then we should first prove the bound on the error introduced by the discretisation method, then prove the error introduced by using floating point numbers instead of real numbers in the program. Then the proof of correctness will be a proof than the result computed by the program at a certain point of the material is in a given interval around the value than could be obtained using a function solution of the partial differential equation. [7,6] follows such an approach for a wave equation and a sequential C program. As the heat diffusion simulation program do not reorder the computations, such a proof of correctness for our heat diffusion simulation should be very similar to the proofs of [7,6] and the proof of the present paper for the parallel aspects. A program that uses the reduce skeleton of OSL would be much more difficult to prove as the reduce produces the same result as a sequential loop if the binary operator is *associative*. However usual operations on floating point numbers *are not* associative.

To be confident that the *compiled* OSL program will be correctly executed, and if we trust the C++ compiler, we could focus on verifying that the *parallel implementation* of the OSL library is correct. We will formalise the *execution model* of the OSL library, ie formalise the implementation of OSL, then we will prove the equivalence of this execution model and the programming model we used in this paper.

Another more ambitious approach is to design a verified compiler. The PaPDAS project (http://traclifo.univ-orleans.fr/PaPDAS) aims at the design of an algorithmic skeleton language extension of C, and at the design, implementation and proof correctness of a compiler for this language (extension of the CompCert compiler [22]). The set of skeletons of this ASC language will be similar to the set of skeletons of OSL. Therefore the proof of correctness of a heat diffusion simulation written in ASC will be very similar to the proof presented here. In addition, with the verified ASC compiler, we would have a proof that the *compiled* version of the heat diffusion simulation program is correct.

Acknowledgements. This work is supported by the *Agence National de la Recherche* and the Japanese Science and Technology agency (French-Japanese project PaPDAS ANR-2010-INTB-0205-02). Noman Javed is supported by a grant from the Higher Education Commission of Pakistan.

References

1. Aldinucci, M., Danelutto, M.: Skeleton-based parallel programming: Functional and parallel semantics in a single shot. Computer Languages, Systems and Structures 33(3-4), 179–192 (2007)
2. Berthold, J., Dieterle, M., Lobachev, O., Loogen, R.: Parallel FFT with Eden Skeletons. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 73–83. Springer, Heidelberg (2009)
3. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Springer, Heidelberg (2004)
4. Bird, R.S.: Lectures on constructive functional programming. In: Broy, M. (ed.) Constructive Methods in Computing Science. NATO ASI, vol. 55. Springer, Marktoberdorf (1989)
5. Bird, R., de Moor, O.: Algebras of Programming. Prentice Hall (1996)
6. Boldo, S.: Floats and Ropes: A Case Study for Formal Numerical Program Verification. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S.E., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 91–102. Springer, Heidelberg (2009)
7. Boldo, S., Clément, F., Filliâtre, J.C., Mayero, M., Melquiond, G., Weis, P.: Formal Proof of a Wave Equation Resolution Scheme: The Method Error. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 147–162. Springer, Heidelberg (2010)
8. Caromel, D., Henrio, L., Leyton, M.: Type safe algorithmic skeletons. In: 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), pp. 45–53. IEEE Computer Society (2008)
9. Ciechanowicz, P., Poldner, M., Kuchen, H.: The Münster Skeleton Library Muesli – A Comprenhensive Overview. Tech. Rep. Working Paper No. 7, European Research Center for Information Systems, University of Münster, Germany (2009)
10. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press (1989)
11. Di Cosmo, R., Pelagatti, S., Li, Z.: A calculus for parallel computations over multidimensional dense arrays. Computer Language Structures and Systems 33(3-4), 82–110 (2007)

12. Dijkstra, E.W.: Letters to the editor: goto statement considered harmful. Comm. of the ACM 11(3), 147–148 (1968)
13. Falcou, J., Sérot, J.: Formal Semantics Applied to the Implementation of a Skeleton-Based Parallel Programming Library. In: Bischof, C.H., Bücker, H.M., Gibbon, P., Joubert, G.R., Lippert, T., Mohr, B., Peters, F.J. (eds.) Parallel Computing: Architectures, Algorithms and Applications, ParCo 2007. Advances in Parallel Computing, vol. 15, pp. 243–252. IOS Press (2007)
14. Falcou, J., Sérot, J., Chateau, T., Lapresté, J.T.: Quaff: Efficient C++ Design for Parallel Skeletons. Parallel Computing 32, 604–615 (2006)
15. Gesbert, L., Hu, Z., Loulergue, F., Matsuzaki, K., Tesson, J.: Systematic Development of Correct Bulk Synchronous Parallel Programs. In: The 11th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 334–340. IEEE Computer Society (2010)
16. González-Vélez, H., Leyton, M.: A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers. Software, Practrice & Experience 40(12), 1135–1160 (2010)
17. Gorlatch, S.: Send-receive considered harmful: Myths and realities of message passing. ACM TOPLAS 26(1), 47–56 (2004)
18. Hidalgo-Herrero, M., Ortega-Mallén, Y.: An Operational Semantics for the Parallel Language Eden. Parallel Processing Letters 12(2), 211–228 (2002)
19. Hu, Z., Takeichi, M., Chin, W.N.: Parallelization in calculational forms. In: POPL 1998: Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 316–328. ACM Press (1998)
20. Javed, N., Loulergue, F.: OSL: Optimized Bulk Synchronous Parallel Skeletons on Distributed Arrays. In: Dou, Y., Gruber, R., Joller, J.M. (eds.) APPT 2009. LNCS, vol. 5737, pp. 436–451. Springer, Heidelberg (2009)
21. Javed, N., Loulergue, F.: A Formal Programming Model of Orléans Skeleton Library. In: Malyshkin, V. (ed.) PaCT 2011. LNCS, vol. 6873, pp. 40–52. Springer, Heidelberg (2011)
22. Leroy, X.: A formally verified compiler back-end. Journal of Automated Reasoning 43(4), 363–446 (2009)
23. Leyton, M., Henrio, L., Piquer, J.M.: Exceptions for Algorithmic Skeletons. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010. LNCS, vol. 6272, pp. 14–25. Springer, Heidelberg (2010)
24. Loulergue, F., Gava, F., Billiet, D.: Bulk Synchronous Parallel ML: Modular Implementation and Performance Prediction. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005, Part II. LNCS, vol. 3515, pp. 1046–1054. Springer, Heidelberg (2005)
25. Matsuzaki, K., Iwasaki, H., Emoto, K., Hu, Z.: A Library of Constructive Skeletons for Sequential Style of Parallel Programming. In: InfoScale 2006: Proceedings of the 1st International Conference on Scalable Information Systems. ACM Press (2006)
26. Tesson, J., Loulergue, F.: A Verified Bulk Synchronous Parallel ML Heat Diffusion Simulation. In: 11th International Conference on Computational Science (ICCS 2011), pp. 36–45. Procedia Computer Science, Elsevier (2011)
27. Valiant, L.G.: A bridging model for parallel computation. Comm. of the ACM 33(8), 103 (1990)

Parallelization of an XML Data Compressor on Multi-cores

Tomasz Müldner[1], Christopher Fry[1],
Tyler Corbin[1], and Jan Krzysztof Miziołek[2]

[1] Jodrey School of Computer Science, Acadia University
Wolfville, B4P 2A9 NS, Canada
{tomasz.muldner,062181f,094568c}@acadiau.ca
[2] IBI AL, University of Warsaw, Warsaw, Poland
jkm@ibi.uw.edu.pl

Abstract. Because of a growing interest in using XML for massive complex data there has been considerable research on designing XML compressors. This paper presents our research aimed at building parallel XML compressors, using Java and OpenMP (with C++). Our findings show that OpenMP is a preferred choice achieving better results than Java using a multi-core platform.

Keywords: XML, XML compression, multi-cores, Java, OpenMP.

1 Introduction

Recent availability of multi-core processors has resurrected interest in parallel programming environments designed to take advantage of multiple threads using shared memory, such as Java [16], TBB [8], and OpenMP [4].

Although XML is now the de facto data format standard for Web services, as well as for encoding semi-structured data, its verbose nature adversely affects various XML services. Given these XML-related performance issues, there has been interest in devising various queryable and updateable XML compressors, *with minimal decompression*.

To the best of our knowledge, to date, no work on applying parallelization techniques to XML processing has been done, which has the potential to improve execution time while maintaining high compression ratio. Our research aims to parallelize XSAQCT (pronounced *exact*), developed by the authors of this paper, which supports querying and updating XML documents, see [15] and [14]. The compression process in our XML compressor XSAQCT involves: (1) encoding the document structure in an *annotated tree*; (2) storing the annotated tree and the document contents in separate *containers*; and (3) applying *back-end compressors* to the containers. Since processing of any XML document starts with parsing, this stage has been recognized as the most important performance **bottleneck**. The disk I/O is much slower than the parallel processing and for various applications, such as an atmospheric model simulation described in [17], the scalability of the system implemented on a multi-core cluster is limited by

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 101–110, 2012.

disk performance operations. We intend to use multi-cores for XML compression in such a way that various cores are used not only to read the XML file and parse it, but also to perform more computation-intensive operations; specifically creating an annotated tree and compressed containers using multiple threads. The main focus of the work described in this paper is to compare the two programming environments, namely Java and OpenMP (with C++) for parallelization of the process of creating compressed containers (steps 2 and 3 in the above description of XSAQCT). This paper is organized as follows. In Section 2, we briefly describe XSAQCT and in Section 3 provide a detailed description of our tests of the two programming environments. In Section 4, we compare results using Java and OpenMP and in Section 5 provide conclusions and describe our future work.

Contributions. First, this paper examines the use of Java parallelization tools applied to creating compressed data containers (as the basic step in process of XML compression). We performed tests on a eight-core platform using five different versions, namely: (1) each similar path is given a container and a thread; (2) a pool of compressing threads; (3) a pool of threads with all compressed data passed to a single writer thread; (4) using multiple writer threads; and finally to better support updates of compressed XML data (5) using multiple writer threads and a database for storage rather than the file system. Second, we repeated all these tests using the same multi-core platform and OpenMP (with C++) rather than Java. The results of our tests show that Java parallelization provides some relative speedup (i.e. a speedup compared to a single-threaded version). Similarly, OpenMP provides some relative speedup. However, comparing timings of both environments, OpenMP turns out to be faster and therefore is a better choice for the task in hand. We show that Version 5 (using a database) is the fastest version and should be used for implementing parallel updateable XML compressors.

2 Brief Description of XSAQCT

Given a document D, we perform a single SAX traversal of D to encode it, thereby creating an annotated tree $T_{A,D}$, in which all *similar paths* (i.e. paths that are identical, possibly with the exception of the last component, which is the data value) are merged into a single path and each node is annotated with a sequence of integers; see Fig. 1. When the annotated tree is being created, data values are output to the appropriate data containers. Next, $T_{A,D}$ is compressed by writing its annotations to one container and the skeleton tree T_D (with annotations stripped) to another container. Finally, all containers are compressed, using back-end compressors (such as GZIP [3] or BZIP2 [1]), and written to create the compressor's output.

The main reason behind using an annotated tree representation is that it can be used to answer various queries as well as to efficiently implement updates. Because of space limitations, here we do not describe the compression technique

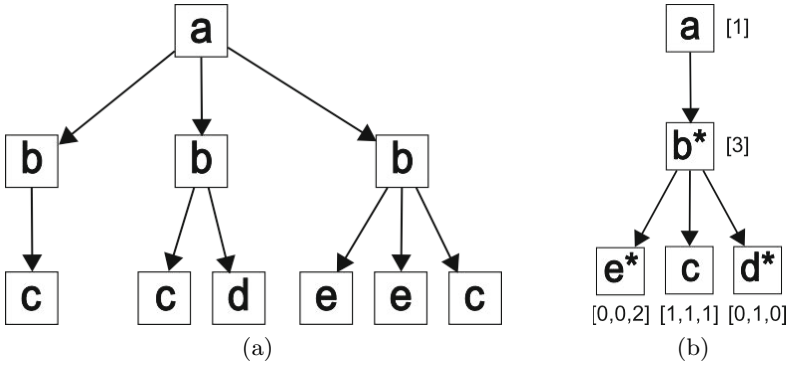


Fig. 1. (a) XML document D; (b) the annotated tree $T_{A,D}$ representing D

used in the presence of cycles, nor do we describe the available update operations (for more details see [14]).

3 Testing the Two Programming Environments

Let us first recall some basic concepts related to parallelization in the context of our application. Speedup S_n is defined as the execution time of the sequential algorithm (a *serial* time, denoted by R) divided by the execution time L_n of the parallel algorithm with n processors, or cores; i.e. $S_n = R/L_n$. Now let O be the *optimal* time for a sequential parser to parse the XML file (with no extra operations, such as compression) and let P be the fraction of time spent on the paralellizable part, such that $P = (R - O)/R$. The *theoretical upper bound* U_n is given by Amdahl's law, we also introduce the *achieved* fraction of time spent on the paralellizable part A_n using the achieved speedup S_n and solving for P the equation, then computing A_n:

$$U_n = \frac{1}{(1-P)+P/n} \qquad S_n = \frac{1}{(1-P)+P/n} \qquad A_n = \frac{n*(1-1/S_n)}{n-1}$$

Efficiency E_n is defined as S_n/n and it indicates whether all processors are well utilized in solving the algorithm.

Since our goal is to test parallel compression of containers, the SAX parser (specifically Xerces [10]), was used to read an XML document from a file and discard all features (such as tags) except text values. Each program was tested on the following three files of varying sizes and the corresponding number of containers: shakespeare.xml (7,894,984 bytes and 40 containers), dblp.xml (133,862,735 bytes and 116 containers) and 1gig.xml (1,172,322,551 bytes and 497 containers). The first two files are taken from the Wratislavia corpus [9], while the last file is a randomly generated XML file, using xmlgen [11]. We tested our code on an eight-core Mac machine running 32-bit Darwin Kernel Version 10.6.0. Specifically, this box uses 2.8GHz Quad-Core Intel Xeon chips (Harpertown/Penryn) processors, and has 12MB of L2 cache per processor. In the parallel versions,

we *always* used eight cores and ran each test three times recording the average. Therefore, showing our results we omit the subscript n, e.g. we use S to denote the speedup.

3.1 Using Java

We used Java 1.6 and implemented five versions of tests. In these tests, all text data associated with a similar path are stored in a single container and compressed, using as a backend compressor Java's *internal* gzip compression library.

Table 1 shows the compression time $L(V_i)$ (in seconds) using five parallel versions denoted by $V_1,\ldots,V_5$ and the serial time R as well as the optimal time O. Table 2 shows the corresponding speedups $S(V_i)$, efficiency $E(V_i)$, parallelizable fractions $P(V_i)$, achieved fractions $A(V_i)$ and upper bounds $U(V_i)$. The detailed discussion of each version is provided in the following sections.

Table 1. Compression time using various Java versions

File	$L(V_5)$	$L(V_4)$	$L(V_3)$	$L(V_2)$	$L(V_1)$	R	O
shakespeare.xml	0.960	0.739	0.698	0.932	0.856	1.242	0.412
dblp.xml	5.518	5.344	5.826	5.758	5.263	10.121	2.999
1gig.xml	30.185	40.678	53.581	54.130	55.260	139.749	10.256

Table 2. Analysis of results using various Java versions

File and analysis	V5	V4	V3	V2	V1
shakespeare.xml					
S(Vi)	1.294	1.681	1.779	1.333	1.451
E(Vi)	0.162	0.210	0.222	0.167	0.181
P(Vi)	0.668	0.668	0.668	0.668	0.668
A(Vi)	0.259	0.463	0.501	0.285	0.355
U(Vi)	2.408	2.408	2.408	2.408	2.408
dblp.xml					
S(Vi)	1.834	1.894	1.737	1.758	1.923
E(Vi)	0.229	0.237	0.217	0.220	0.240
P(Vi)	0.704	0.704	0.704	0.704	0.704
A(Vi)	0.520	0.539	0.485	0.493	0.549
U(Vi)	2.602	2.602	2.602	2.602	2.602

Continued on next page

Table 2. (*Continued*)

File and analysis	V5	V4	V3	V2	V1
1gig.xml					
S(Vi)	4.630	3.435	2.608	2.582	2.529
E(Vi)	0.579	0.429	0.326	0.323	0.316
P(Vi)	0.927	0.927	0.927	0.927	0.927
A(Vi)	0.896	0.810	0.705	0.700	0.691
U(Vi)	5.285	5.285	5.285	5.285	5.285

Version 1. In the first version, each similar path is given a container and a thread (therefore the number of containers is the same as the number of threads); see Fig. 2. Note that Version 1 cannot be scaled to fewer than the maximum number of available cores (which in this case is eight).

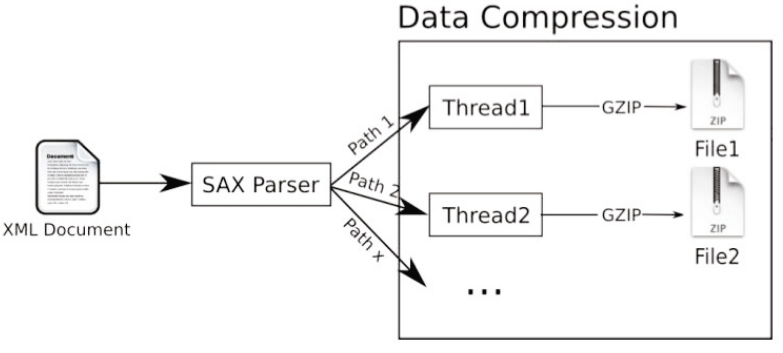


Fig. 2. Overview of the first version

In Table 1, column L(V_1) we show the compression times for all three files running Version 1. Table 2, column V_1 shows some speedup, but it was much less than the theoretical upper bound. For example, for shakespeare.xml the speedup was 1.451 while the theoretical upper bound was 2.408. Our hypothesis stating that the presence of a large number of threads running reduces the advantage of having multiple cores was supported by another test, namely running Version 1 on enwikinews.xml file from the XML corpus, where there are 116 threads and there is no speedup resulting from using a parallel version. In this file, 86% of text data belongs to a single similar path, while each other path has less than 5% of the total text data. Since we don't know in advance (before the parsing is completed) the size of each container, a better solution may be to improve load balancing by giving each thread approximately the same amount of data to compress.

Version 2. In the second version, we use a pool of compressing threads and the user can choose the size of the pool thereby using a specific number of cores. Here, the parser thread passes data to the next compressing thread in the pool. Therefore, any thread can compress data in any container and as a result the load is more balanced. However, if no threads are available, or if threads are given too much work, then the parser thread is forced to wait. Table 1, column L(V_2) provides the compression times for this version and shows that they are much better than using the serial version, although worse than using the optimal version. Corresponding analysis provided in Table 2 shows that speedups, efficiency and achieved parallelizable fractions in Version 2 are roughly the same as in Version 1. Since in this version the number of cores used can be selected, we tried to run our program using two, four, six and eight cores, but found that while moving from two to four cores the compression time gets better, moving beyond four cores does not improve the speedup. One possible reason for the lack of speedup is that even though many threads can compress in parallel, they must wait for an exclusive lock on a GZIPOutputStream to *start* compressing (and then write to an output file). Therefore, for a similar path that holds large amount of data, there may be many threads waiting to get the *lock*. Unfortunately, compressing *in memory* is not feasible as too much data may reside in memory, and in addition we must ensure compressed data is written in the correct order. Our next version uses many compressors and a single writer.

Version 3. The third version is similar to the second version and uses a pool of threads, but all compressed data is passed to a *writer thread* thereby avoiding waiting for a lock. In addition, each container stores a buffer and an integer *order value*. When a buffer is full, it is passed to an available compressor thread from the pool. Compressor threads can compress in parallel without writing to a file because the order number is used to determine the correct order in which buffers are written. Table 1, column L(V_3) provides the compression times for this version and shows no significant improvement over Version 2. Corresponding results from Table 2 show that there is a slightly better speedup, efficiency and achieved parallelizable fractions for the smallest file, shakespeare.xml, but other results are roughly the same as in the previous versions. This may indicate that the writer thread receives data faster than it can write it.

Version 4. In this version, we used *multiple writers* and when data needs to be compressed, the next thread in the pool is used and when a writing operation is performed, the next writer thread is used. Table 1, column L(V_4) provides the compression times for this version with *two* writer threads, and shows that the compression time is significantly better only for the largest file, 1gig.xml. Corresponding results from Table 2 show that other results are roughly the same as in the previous version. This may indicate that the writer threads contend with each other for access to a file output. Adding additional writers does not result in any improvement.

Version 5. The final version uses multiple worker threads described in the previous version and additionally it tries to remove limitations imposed by the file system (in particular pertaining to updates on compressed XML files) by using Oracle's Berkeley DB Java Edition 11.2.4.0 [5]. Table 1, column L(V_5) provides the compression times and again show that the compression time is significantly better only for the largest file, 1gig.xml. Corresponding results from Table 2 show a speedup of 4.63 for the largest file, i.e. 1gig.xml using Version 5, which is better than using Version 4 and approaching the theoretical speedup of 5.285, but other results are roughly the same or even worse than in the previous versions. It appears that the drawback of using a database is that it uses its own threads to perform logging, cache control and database cleanup.

Conclusion. In comparing five versions using Java, we determined that for 1gig.xml each version is better than its predecessor, with Version 5 - using the database backend - achieving the greatest speedup. Another desirable property is that the larger the XML file the greater the relative speedup and this occurs in all versions except Version 3.

3.2 Using OpenMP/C++

We used OpenMP with C++ on the same platform as Java using the Boost GZip Filters [2] for compression. However, one slight alteration to our design requirements comes from the fact that OpenMP is *"'. . . based upon the existence of multiple threads in the shared memory programming paradigm. . . "* [12], thus defining OpenMP as a more "task" based paradigm rather than Java, which is a "thread" based paradigm. Specifically the design limitations are seen in Version 1 (see Section 3.1.1) where the OpenMP specifications are not designed to handle a dynamically increasing number of threads. To overcome this problem and to gauge some type of performance, each similar path is given a container and POSIX thread [13]. Tables 3 and 4 correspond respectively to Tables 1 and 2, except they show results using OpenMP.

Table 3 shows that using OpenMP/C++ parallel implementations are much faster than the serial implementations. In particular, parallel Version 1 is 58% faster than the serial version for the largest file, 1gig.xml. It also shows that using Version 2 (data is passed by the parser thread to the next compressing thread, see Section 3.1.2) and comparing it to Version 1, there was a only slight increase in performance for the two first two files, but a 140% increase for the largest file. Version 3, in which all compressed data is passed to a writer thread, see Section 3.1.3, provides an incremental increase over Version 2, for the first two files and little increase for the third file. Finally, both Version 4 (separate compressing and multiple-writer threads, see Section 3.1.4) and Version 5 (using a database, see Section 3.1.5) exhibit only a little speedup, if any for the first two files but larger speedup for the third, the largest file. In particular, for this file and Version 5 the speedup is 3.022 while the theoretical speedup is 3.705. Lack of larger speedup can be attributed to the overhead in creating a database backend being the limiting factor.

Table 3. Compression time using various OpenMP versions

File	L(V_5)	L(V_4)	L(V_3)	L(V_2)	L(V_1)	R	O
shakespeare.xml	0.391	0.280	0.282	0.539	0.555	0.512	0.116
dblp.xml	4.547	4.718	4.810	5.149	5.317	6.950	2.259
1gig.xml	25.599	31.910	31.815	31.943	44.987	77.353	12.812

Table 4. Analysis of results using various OpenMP versions

File and analysis	V5	V4	V3	V2	V1
shakespeare.xml					
S(Vi)	1.309	1.829	1.816	0.950	0.923
E(Vi)	0.164	0.229	0.227	0.119	0.115
P(Vi)	0.773	0.773	0.773	0.773	0.773
A(Vi)	0.270	0.518	0.513	-0.060	-0.096
U(Vi)	3.094	3.094	3.094	3.094	3.094
dblp.xml					
S(Vi)	1.528	1.473	1.445	1.350	1.307
E(Vi)	0.191	0.184	0.181	0.169	0.163
P(Vi)	0.675	0.675	0.675	0.675	0.675
A(Vi)	0.395	0.367	0.352	0.296	0.269
U(Vi)	2.443	2.443	2.443	2.443	2.443
1gig.xml					
S(Vi)	3.022	2.424	2.431	2.422	1.719
E(Vi)	0.378	0.303	0.304	0.303	0.215
P(Vi)	0.834	0.834	0.834	0.834	0.834
A(Vi)	0.765	0.671	0.673	0.671	0.478
U(Vi)	3.705	3.705	3.705	3.705	3.705

The most interesting findings are that the size and structure of the input file determines the results. Specifically, for shakespeare.xml and dblp.xml results are better using versions 3 and 4 than using versions 1 and 2, while for 1gig.xml the best results are using Version 5.

4 Comparison of Java and OpenMP/C++

In this section we compare the speeds of using OpenMP comparing to Java, i.e. the ratios of the OpenMP versions to their Java counterparts, see Fig. 3, where in column labels, the suffix "-J" indicate the Java versions.

To avoid any confusion, let us stress that Java's single threaded implementations are much slower than the corresponding OpenMP's implementation. Comparing parallelizable part for the largest file, 1gig.xml, it is 0.927 for Java and 0.834 in OpenMP/C++. Looking at speedups it may appear that Java benefits the most from using multiple cores, but comparing timing results and considering parallelizable parts shows that OpenMP is superior for all three files. Specifically, for shakespeare.xml, the smallest XML file, OpenMP is significantly better than Java both in terms of relative speedup and time taken to process the file.

For dblp.xml, the next largest file, the Java implementation benefited the most from having multiple cores available, as its relative speed up was moderately greater than OpenMP. However considering all factors OpenMP is better than using Java.

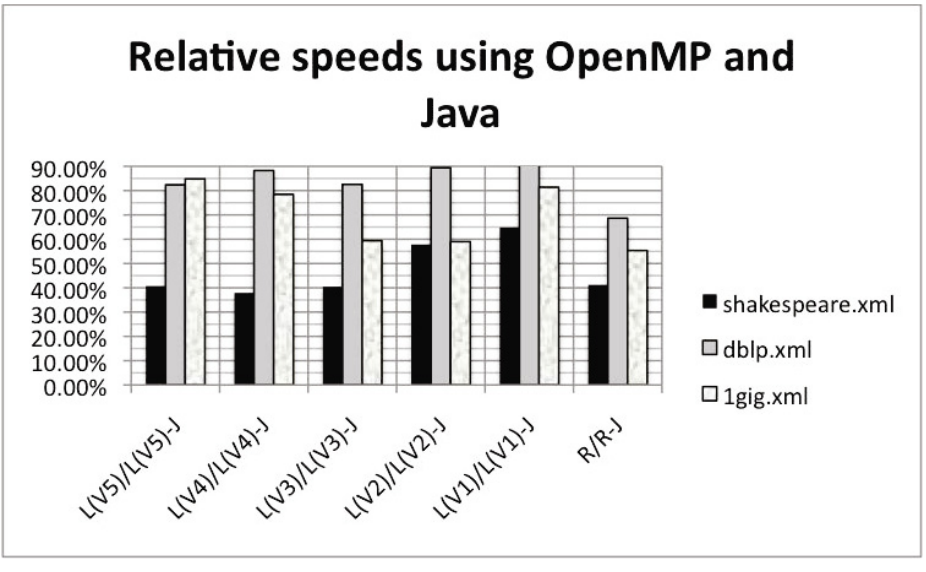


Fig. 3. Comparison of Java and OpenMP

5 Conclusions and Future Work

In this paper, we described our work on using multi-core machines for XML data compression, with Java and OpenMP. Our tests show that while both platforms have showed potential to increase performance and productivity, OpenMP is much better suited for the task in hand, and it makes it possible to achieve considerable gains using eight cores when compared to using a single core. In addition, our tests show the advantage of using Version 5, with multiple compressors and a database backend, which is the preferred choice for all update operations on compressed XML files.

The alternative to using internal gzip as a *backend* compressor would be to use already existing external *parallel compressor*, such as pigz [7] or pbzip2 [6]. Finally, we will test our algorithms using TBB, run all algorithms on various platforms and design and implement parallel algorithms for *other stages* of XSAQCT.

Acknowledgments. We would like to thank Dr. Darcy Benoit for letting us use his eight core Mac for our tests. We would also like to thank the anonymous reviewers for their detailed comments, which were very helpful to improve the final version of our paper.

References

1. bzip2, `http://www.bzip.org/` (retrieved on April 20, 2010)
2. C++ library: boost, `http://www.boost.org/users/news/version_1_45_0` (retrieved on April 20, 2010)
3. gzip compression, `http://www.gzip.org/` (retrieved on April 20, 2010)
4. OpenMP, `http://openmp.org/wp/about-openmp/` (retrieved on April 20, 2010)
5. Oracle Berkeley DB 11g, `http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html` (retrieved on April 20, 2010)
6. Parallel BZIP2, `http://compression.ca/pbzip2` (retrieved on April 20, 2010)
7. pigz - parallel gzip, `http://www.zlib.net/pigz` (retrieved on April 20, 2010)
8. TBB, `http://www.threadingbuildingblocks.org/` (retrieved on April 20, 2010)
9. Wratislavia XML corpus, `http://www.ii.uni.wroc.pl/~inikep/research/Wratislavia/` (retrieved on April 20, 2010)
10. Xerces, `http://xerces.apache.org/xerces-j/` (retrieved on April 20, 2010)
11. xmlgen - the benchmark data generator, `http://www.xml-benchmark.org/generator.html` (retrieved on April 20, 2010)
12. Barney, B.: OpenMP Tutorial, `https://computing.llnl.gov/tutorials/openMP/` (retrieved on April 20, 2010)
13. Butenhof, D.: Programming with POSIX threads. Addison-Wesley professional computing series. Addison-Wesley (1997), `http://books.google.ca/books?id=_xvnuFzo7q0C`
14. Müldner, T., Fry, C., Miziołek, J., Corbin, T.: Updates of compressed dynamic xml documents. In: Eight International Network Conference, INC 2010, Heidelberg, Germany, pp. 315–324 (July 2010)
15. Müldner, T., Fry, C., Miziołek, J., Durno, S.: XSAQCT: XML queryable compressor. Balisage: The Markup Conference, Montréal, Canada (August 2009), `http://www.balisage.net/Proceedings/vol3/html/Muldner01/BalisageVol3-Muldner01.html`
16. Oaks, S., Wong, H.: Java Threads, 3rd edn. O'Reilly Media, Inc. (2004)
17. Osthoff, C., Schepke, C., Panetta, J., Grunmann, P., Maillard, N., Navaux, P., Dias, P.L.S., Lopes, P.P.: I/O Performance Evaluation on Multicore Clusters with Atmospheric Model Environment. In: Proceedings of the 2010 22nd International Symposium on Computer Architecture and High Performance Computing Workshops, SBAC-PADW 2010, pp. 49–54. IEEE Computer Society, Washington, DC (2010), `http://dx.doi.org/10.1109/SBAC-PADW.2010.15`

Comparing CUDA, OpenCL and OpenGL Implementations of the Cardiac Monodomain Equations

Rafael Sachetto Oliveira[1,2], Bernardo Martins Rocha[3,4], Ronan Mendona Amorim[3], Fernando Otaviano Campos[5], Wagner Meira Jr.[2], Elson Magalhes Toledo[4], and Rodrigo Weber dos Santos[3]

[1] Federal University of São João Del Rey, Department of Computer Science
[2] Federal University of Minas Gerais Department of Computer Science
[3] Federal University of Juiz de Fora, Department of Computer Science and Computational Modeling Program,
[4] National Laboratory of Scientific Computing
[5] Medical University of Graz, Institute of Biophysics

Abstract. Computer simulations of cardiac electrophysiology are a helpful tool in the study of bioelectric activity of the heart. The cardiac monodomain model comprises a nonlinear system of partial differential equations and its numerical solution represents a very intensive computational task due to the required fine spatial and temporal resolution. Recent studies have shown that the use of GPU as a general purpose processor can greatly improve the performance of simulations. The aim of this work is to study the performance of different GPU programming interfaces for the solution of the cardiac monodomain equations. Three different GPU implementations are compared, OpenGL, NVIDIA CUDA and OpenCL, to a CPU multicore implementation that uses OpenMP. The OpenGL approach showed to be the fastest with a speedup of 446 (compared to the multicore implementation) for the solution of the nonlinear system of ordinary differential equations (ODEs) associated to the solution of the cardiac model, whereas CUDA was the fastest for the numerical solution of the parabolic partial differential equation with a speedup of 8. Although OpenCL provides code portability between different accelerators, the OpenCL version was slower for the solution of the parabolic equation and as fast as CUDA for the solution of the system of ODEs, showing to be a portable way of programming scientific applications but not as efficient as CUDA when running on Nvidia GPUs.

1 Introduction

Simulations of cardiac electrophysiology have become a valuable tool for the study and comprehension of heart's bioelectric activity under normal and pathological conditions. These simulations are usually based on the bidomain model [5], which is a system of two Partial Differential Equations (PDEs) coupled to a set of nonlinear Ordinary Differential Equations (ODEs) describing the behavior of

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 111–120, 2012.

the membrane of cardiac cells. In spite of being currently the most complete description of the electrical activity of the heart, the numerical solution of the bidomain equations is a computational challenging task [8]. Usually the bidomain equations are reduced to the simpler monodomain model by considering that: the extracellular potential is constant; or tissue conductivity is isotropic; or the intracellular and extracellular conductivities have equal anisotropy ratios [5].

Nevertheless, numerical approximation of these models involves the spatial and temporal discretization of the PDEs as well as the integration of the nonlinear systems of ODEs. Using an appropriate decomposition for the time discretization, the nonlinearity of the system may be isolated, i.e. for each node of the spatial mesh a system of ODEs, that comes from the cellular model, may be solved independently. To perform realistic simulations of cardiac tissue, a large number of ODE systems must be solved at each time step, which contributes substantially to the total computational work. Therefore, it is necessary to pursue new efficient ways of solving the large linear algebraic system that arises from the discretization of the PDEs as well as the systems of ODEs associated with the cell models.

The use of the graphics processing units (GPUs) for general purpose (GPGPU) programming is becoming more popular and there are different approaches for this kind of parallel programming. Currently, the most popular environment for GPU programming is the NVIDIA CUDA parallel computing architecture. Recent studies have reported significant performance gains obtained using NVIDIA CUDA to parallelize the solution of the monodomain model [3,9].

The aim of this work is to compare the performance of different approaches for GPU programming to solve the monodomain problem. We present a comparison between different implementations using OpenGL, NVIDIA CUDA and OpenCL for the numerical solution of the monodomain equations on modern hardware platforms.

2 Governing Equations and Numerical Scheme

2.1 Monodomain Model

In cardiac tissue, the excitation wave spreads through the tissue because the cardiac cells are electrically coupled via special proteins called gap junctions. This phenomenon is mathematically described by a reaction–diffusion equation referred to as the monodomain equation, given by

$$\beta C_m \frac{\partial V_m}{\partial t} + \beta I_{ion}(V_m, \boldsymbol{\eta}) = \nabla \cdot (\boldsymbol{\sigma_m} \nabla V_m) + I_{stim} \tag{1}$$

$$\frac{\partial \boldsymbol{\eta}}{\partial t} = \boldsymbol{f}(V_m, \boldsymbol{\eta}) \tag{2}$$

where β is the surface-to-volume ratio of the cardiac cells, C_m is the membrane capacitance, V_m is the transmembrane voltage, I_{ion} is the density of the total ionic current which is a function of V_m and a vector of state variables $\boldsymbol{\eta}$, I_{stim} is a stimulus current, $\boldsymbol{\sigma_m}$ is the monodomain conductivity tensor and $\boldsymbol{f}$ is a

vector-valued function of the transmembrane voltage and state variables vector. The tissue is assumed to be isolated along its boundaries, i.e. no–flux boundary conditions are imposed on V_m along all myocardial surfaces.

In this work the classical Luo–Rudy I (LRI) model [2] that describes the electrical activity in a general mammalian ventricular cell was considered to simulate the kinetics of I_{ion} in Eq. (2). In this mammalian ventricular model, I_{ion} is defined as the following sum of currents:

$$I_{ion} = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b \tag{3}$$

where I_{Na} is the fast sodium current, I_{si} is the slow inward current, I_K is the time-dependent potassium current, I_{K1} is the time-independent potassium current, I_{Kp} is the plateau potassium current and I_b is time-independent background current. The LRI model is based on a set of 8 ODEs describing ionic currents and intracellular calcium concentration [2].

2.2 Numerical Scheme

The reaction and diffusion part of the monodomain equations can be split by employing the Godunov operator splitting [5]. Each time step involves the solution of two different problems: a nonlinear system of ODEs

$$\frac{\partial V_m}{\partial t} = \frac{1}{C_m}\left[-I_{ion}(V_m, \eta_i)\right] \tag{4}$$

$$\frac{\partial \eta_i}{\partial t} = f(V_m, \eta_i) \tag{5}$$

and a parabolic linear PDE

$$\frac{\partial V_m}{\partial t} = \frac{1}{\beta C_m}\left[\nabla \cdot (\boldsymbol{\sigma} \nabla V_m)\right] + I_{stim} \tag{6}$$

To obtain a numerical solution of the parabolic equation (6), first, the standard Galerkin Finite Element Method (FEM) was employed for the spatial discretization and then, the second–order Crank-Nicolson (CN) time stepping method was used to obtain a fully discrete approximation. Using the FEM with bilinear elements results in a sparse linear system of equations that needs to be solved at each time step and in the particular case of the CN method has the following form:

$$(M + \tfrac{1}{2}cK)\upsilon^{k+1} = (M - \tfrac{1}{2}cK)\upsilon^k \tag{7}$$

where υ discretizes V_m at time $k\Delta t$; M and K are the mass and stiffness matrices from the FEM discretization, respectively; and the constant c is $\frac{\Delta t}{\beta C_m}$. The linear system was solved by employing the Jacobi preconditioner with an iterative Conjugate Gradient (PCG) solver.

The systems of ODEs in (4)-(5) were integrated using the explicit Euler method. Although it is well known that explicit numerical methods have strong limitations because of stability and accuracy restrictions [6], they are widely used in cardiac simulations due to their simplicity of implementation [4].

2.3 Sparse Matrices Storage

In the present work only 2D regular square meshes were considered, resulting in diagonally structured matrices with 9 diagonals. In order to efficiently store the matrices the following two sparse matrix formats were used: the *compressed sparse row* (CSR) format and the *diagonal* (DIA) format [11,7]. The CSR format was used to assemble the finite element matrices and then the simulations were performed either using the CSR or DIA format.

Let N and Nz be the number of rows of the matrix and the total number of non-zero entries of the matrix, respectively. Then, the CSR format stores a sparse matrix in three arrays: `vals` and `cols` which are both of size Nz and hold, respectively, the non-zero entries and the columns indexes of these entries and finally, a third array named `ptrs` of size $N + 1$ that stores pointers to the beginning of each row in `vals` and `cols`.

The diagonal format uses 2 arrays to store the sparse matrix: the first is a rectangular array of $N \times Nd$ entries, where Nd is the number of diagonals while the second, `offsets` is a one–dimensional array that stores the diagonal offset with respect to the main diagonal.

3 Implementations

The code that implements the numerical solution of the monodomain problem (2) using the LRI cardiac cell model was written in ANSI C and parallelized using OpenMP. The GPU implementations were all extended from the CPU code to the different GPU computing environments, namely: OpenGL, NVIDIA C for CUDA and OpenCL. Hence, particular functions were developed for each environment to solve: (i) the system of ODEs and (ii) the parabolic problem. For the ODE problem the explicit Euler method was developed, whereas for the parabolic problem the building blocks (vector operations such as SAXPY, dot product and sparse matrix vector multiplication) of the PCG were written for each GPU code. The details of each GPU implementation are described below.

3.1 OpenGL Implementation

This implementation uses the same programming resources available for graphics computing, therefore a better understanding of this approach requires a background knowledge in computer graphics. The main idea behind using OpenGL for GPGPU computation is to map some of the computer graphics concepts into CPU programming concepts.

Basically, in computer graphics, textures are used as input data for rendering, which draws to the frame buffer. So in our case this machine of rendering graphics is used to perform general purpose computation. Therefore, for our GPGPU approach, the textures were used to provide the input data necessary to the computation, while the rendering process performed the computation that produced the results which were then written to the frame buffer.

The graphics hardware is implemented as a pipeline, and, in recent GPUs, there are some stages of this pipeline that can be programmed. For example, the vertex processor and the fragment processor. We used the fragment processor in our implementation because it fits better to the problem and it is the most powerful processor on the GPU. The fragment processor executes a fragment shader, that is a program written to the fragment processor. The fragment shader can be written using *shading languages* such as the OpenGL Shading Language(GLSL) and is responsible for calculating the color of individual pixels. The shader can only write to its own position, or in other words, scatter writes are not possible. Textures, however, can be randomly accessed although it is better to keep the access with a certain locality because of the texture cache.

There are much more details involved in this complex task which we do not discuss here. For further discussion on the OpenGL implementation see [1].

3.2 GPU Implementation Using NVIDIA C for CUDA

The CUDA parallel programming model extends the C language with a minimalist set of abstractions for expressing parallelism. That is, it lets the programmer focus on important issues of parallelism rather than dealing with unfamiliar and complicated concepts from computer graphics (as is the case when using OpenGL for GPU programming) in order to explore the computational power of GPUs for general purpose computation.

A CUDA program is organized into a host program running on the CPU and one or more parallel special C functions called *kernels* that are suitable for execution on a parallel processing device like the GPU. A kernel executes a scalar sequential program on a set of parallel threads. The threads are organized into a grid of thread blocks by the programmer. All the threads within a single thread block are allowed to synchronize with each other via barriers and have access to a high–speed, per–block shared memory which allows inter–thread communication. Threads from different blocks in the same grid can coordinate only via operations in a global memory visible to all threads. CUDA requires thread blocks to be independent, meaning that a kernel must execute correctly no matter the order in which blocks are scheduled to run.

The NVIDIA CUDA solver of the cardiac monodomain equations was previously presented in [9], where details of the implementation are discussed.

3.3 GPU Implementation Using OpenCL

OpenCL is an open standard that can be used to program CPUs, GPUs, and other devices from different vendors. Our OpenCL solver was based in the previously CUDA monodomain solver presented in [9]. We relied on a previously implemented code mainly to make the comparisons fair. In addition, converting a kernel that uses C for CUDA to an OpenCL kernel involves minor changes.

The major portion of code that had to be rewritten was the host one, for example to detect and setup the GPU or to copy data between host and device. It its important to point out that there are minor differences between NVIDIA

and AMD host codes. Table 1 shows some of the changes we had to make to the CUDA kernel code in order for it to compile and run under OpenCL.

Table 1. Kernel directives in CUDA C and OpenCL

Change	CUDA kernel	OpenCL Kernel
Type qualifiers	__shared__, __global__,etc.	__local, __global.
GPU thread indexing	threadIdx, blockIdx, etc	get_local_id(), get_global_id(), etc
Thread synchronizing	__syncthreads()	barrier()
Kernel declaration	__global__ void...	__kernel void...

4 Methods

The test cases for benchmarking code performance consisted of three different *in-silico* tissue preparations in which the fibers were set along the longitudinal direction and the spatial discretization was set to $\Delta x = 50\,\mu\,m$. The simulations were carried out for 200 ms, after an initial current stimulus, using a time step of $\Delta t = 0.01\,ms$, which guarantees the stability of the explicit Euler scheme for the LRI model.

Table 2 presents the dimensions of the simulated tissues as well as the properties of the FEM meshes. The relative tolerance of the PCG method was set to 10^{-8}. The parameters of the monodomain model were taken from the literature [5]: $C_m = 1\mu F/cm^2$; $\beta = 2000\ cm^{-1}$. Tissue conductivity was considered homogeneous with the monodomain conductivity tensor entries given by $\sigma_l = 3.0\ mS/cm$, $\sigma_t = 1.0\ mS/cm$ and $\sigma_n = 0.31525\ mS/cm$.

Table 2. Details of the *in-silico* tissue preparation and properties of the meshes

Tissue	Nodes	Elements	Nonzero
$10\ mm \times 10\ mm$	200×200	39601	357604
$20\ mm \times 20\ mm$	400×400	159201	1435204
$40\ mm \times 40\ mm$	800×800	638401	5750404

4.1 The ODE and Parabolic Problems

The state variables associated with the systems of ODEs of m cells were stored in a unidimensional vector of size mn called `SV`, where n is the number of differential equations of the ionic model ($n = 8$ in the LRI model [2]). In the CPU implementation the state variables of the cells are stored in `SV` following a row–major ordering, that is, the first 8 entries of `SV` are the state variables associated with the first cell. During the solution phase of the problem a vector containing the V_m of each node has to be passed as argument for the PCG method. In order to avoid extra memory transactions on the GPU implementations, we rearranged the `SV` array using a column–major ordering, where the first m entries of this array represent the V_m values of each node. The GPU implementation of the time stepping of the ODEs uses one thread to solve one ODE system, that is, each thread is associated with each node.

The solution of the parabolic problem consists of one sparse matrix vector multiplication and the solution of a system of linear equations, which also requires basic linear algebra operations such as vector scalar product and SAXPY operations. The GPU implementation of these linear algebra building blocks is a straightforward task since each thread (or work item in OpenCL terminology) is assigned to do the computation of one position of the array, whereas for the scalar product a reduction is required. The computation of the sparse matrix vector multiplications $y = Ax$ using either the CSR format or the DIA format were performed assigning one matrix row to each thread, which then computed one entry of the output vector y.

4.2 Numerical Error

To ensure that there were no deviations in the numerical solutions due to the use of single precision and compiler optimization flags, we compared the results against reference solutions obtained on the CPU using double precision floating point arithmetic. We used the relative root–mean–square (RRMS) as a measure for the error:

$$e = 100 \frac{\sqrt{\sum_t \sum_{i,j} (v_{i,j} - \nu_{i,j})^2}}{\sqrt{\sum_t \sum_{i,j} (v_{i,j})^2}} \tag{8}$$

where v is the reference solution, ν is the numerical solution computed by the GPU that we want to check and i, j and t are indexes in space and time, respectively.

5 Results

In this section we present the numerical results obtained for the monodomain simulations using a ventricular cell model on graphics processing units through OpenGL, NVIDIA CUDA and OpenCL environments. Simulations were carried out on a quad-core Intel Core i7 860 2.80GHz, 8GB of memory equipped with: (i) NVIDIA GeForce GTX 470 with a total of 448 Cuda cores, 1 GB GDDR5 of memory and 133.9 GB/s of memory bandwidth or (ii) AMD Radeon 6850 with 960 Stream processors, 1 GB GDDR5 of memory and 128 GB/s of memory bandwidth.

All the host code was compiled with GNU C compiler version 4.4.5 using the `-O3` compiler flag. The CUDA version was developed with CUDA Toolkit v3.2 and the OpenCL code was developed with NVIDIA OpenCL v1.0 and AMD APP SDK V2.4 with OpenCL 1.1 support. For each test we distinguish between two versions of the GPU code: a version without compiler optimizations and a version with optimizations. The only exception is the OpenGL implementation which was treated as CPU code and therefore compiled only with the `-O3` flag. The compilation flags used for CUDA and OpenCL are different. However, they were used in order to optimize the code as much as possible with respect to mathematical operations. We notice here that the ODE part of the problem has a substantial volume of mathematical operations that are affected by these compilation

flags. For the CUDA implementation the `-use_fast_math` flag was used, while the following flags were used for the OpenCL version: `-cl-fast-relaxed-math`, `-cl-mad-enable` and `-cl-no-signed-zeros`.

The performance of the OpenMP multi–core code over a single core version was previously reported in [9]. There, speedups of 3.98 and 2.35 (comparing 4 cores against single core) were achieved for the ODE and parabolic problems, respectively. In the next tables the OpenMP entry accounts for the performance results obtained with the CPU implementation running with 4 processing cores.

Table 3 gives the elapsed time required to integrate the ODE problem. We observed that for all problem sizes, the OpenGL code performed better than both OpenCL and CUDA versions with or without optimizations. Further, we noticed that turning on the mathematical optimization flags makes the OpenCL (and also the CUDA) version noticeably more effective than without optimization. The tables entries marked with * represent the cases that could not be simulated due to portability of the code. It reflects the fact that NVIDIA CUDA does not run on AMD graphics card and our OpenGL code was implemented using features (textures) that are not supported by AMD GPU hardware.

Table 4 describes the performance of the GPU solvers for the parabolic problem, which mainly depends on the performance of sparse matrix vector multiplication operations, which in turn depends on the storage format used. It is clear that for this part of the problem the computational gain with respect to the multi–core CPU version is smaller than for the ODE part, which is considered to be an embarrassingly parallel computational problem. In this case the CUDA implementation without and with optimizations (*-opt) outperformed the other two versions using both CSR and DIA sparse matrix formats. At this point it is important to recall that the CUDA code used textures in order to improve the performance when fetching data from memory and that OpenCL does not support this feature.

The results show that, for the ODE problem with the largest tissue preparation, GPU speedups (GPU version compared to OpenMP with 4 cores) of 277, 286 and 446 were obtained for the OpenCL, CUDA and OpenGL, respectively, when using the NVIDIA GTX 470 hardware. A GPU speedup of 107 was achieved for the OpenCL code running on AMD Radeon 6850. GPU speedups of about 4–5 were observed for the parabolic problem using the DIA sparse matrix format for both OpenCL and OpenGL. Better performance was achieved for the CUDA version where in the largest problem the code was 8 times faster. A more modest GPU speedup of 2 was obtained when using the AMD Radeon 6850 hardware.

Finally, we computed the numerical error against reference solutions which were carried out using the CPU code with double precision floating point arithmetic. The errors for the OpenCL and CUDA implementations, with an without optimization flags, were smaller than 0.0050%. The errors computed for the OpenGL implementation were about 0.011 %. The results suggested that there were no deviations in the solution neither by use of single precision nor by the use of the several optimization flags used for compilation.

Table 3. Comparison between CPU and GPU solvers for the ODE problem. Execution times are given in seconds.

	GPU	GeForce GTX 470		AMD Radeon 6850	
	Problem	ODE	ODE-opt	ODE	ODE-opt
200 × 200	OpenMP	285.65	285.65	285.65	285.65
	OpenCL	3.39	2.32	4.99	4.99
	OpenGL	2.86	2.62	*	*
	CUDA	3.57	1.63	*	*
400 × 400	OpenMP	1145.63	1145.63	1145.63	1145.63
	OpenCL	9.21	5.09	12.54	12.48
	OpenGL	4.11	4.09	*	*
	CUDA	11.59	4.51	*	*
800 × 800	OpenMP	4588.23	4588.23	4588.23	4588.23
	OpenCL	33.74	16.55	42.85	42.85
	OpenGL	10.21	10.21	*	*
	CUDA	42.59	16.03	*	*

Table 4. Comparison between CPU and GPU solvers for the parabolic problem using either the CSR or the DIA matrix format. Execution times are given in seconds.

	Version	GeForce GTX 470			
	Problem	CSR	CSR-opt	DIA	DIA-opt
200 × 200	OpenMP	43.02	43.02	*	*
	OpenCL	137.79	137.79	91.52	88.26
	OpenGL	216.38	205.69	135.63	135.63
	CUDA	66.03	65.47	20.64	20.64
400 × 400	OpenMP	254.03	254.03	*	*
	OpenCL	305.15	305.15	116.70	116.70
	OpenGL	505.45	503.84	173.42	172.61
	CUDA	225.79	225.44	47.32	47.32
800 × 800	OpenMP	1285.11	1285.11	*	*
	OpenCL	971.64	970.47	263.88	263.88
	OpenGL	1562.95	1562.95	312.52	312.52
	CUDA	850.21	850.21	149.16	149.16
	Version	AMD Radeon 6850			
200 × 200	OpenCL	272.93	272.93	246.27	235.72
400 × 400	OpenCL	477.79	477.12	311.63	306.45
800 × 800	OpenCL	1242.22	1242.22	589.24	589.24

6 Discussion

We have evaluated the performance of different GPU solvers for the monodomain model using OpenGL, NVIDIA CUDA and OpenCL parallel environments. Three different *in-silico* tissue preparations were used in this work for the performance tests. We have shown that in all cases, considering the whole simulation, that the GPU outperformed the parallel multi–core CPU implementation using OpenMP. Also shown is the fact that the GPU performance depended on the GPU programming language adopted. For instance, in the ODE problem the OpenGL code delivered an excellent performance, even when compared to the other two GPU approaches studied in this work. However, the OpenGL approach is hard to program, maintain and understand due to the complexity in mapping the problem concepts to computer graphics concepts for GPGPU computation. It is also shown that for the parabolic problem the CUDA code was the most effective between the GPU solvers.

The OpenCL code was slower than the CUDA version for the solution of the parabolic PDE, where a performance overhead of 77% was observed for the OpenCL code. On the other hand, for the solution of the system of ODEs, the performance of OpenCL and CUDA were very similar. Although OpenCL provides application portability and be can run on different accelerator devices (such

as GPUs, multicore and Cell processors), it was shown that it induces a significant performance cost when running on Nvidia GPUs in comparison to code written in CUDA.

6.1 Related Work

Recently, Weber et al. have presented the performance and programmability of several accelerators running a Quantum Monte Carlo application in [10]. The study compared performance on several platforms including CUDA, Brook+ with ATI graphics accelerators, OpenCL running on both multicore and graphics processors and more. In agreement the results of this study, it was reported that although OpenCL provides application portability, it induces a performance cost.

References

1. Amorim, R.M., Haase, G., Liebmann, M., dos Santos, R.W.: Comparing CUDA and OpenGL implementations for a Jacobi iteration. In: International Conference on High Performance Computing & Simulation (HPCS 2009). pp. 22–32 (2009)
2. Luo, C., Rudy, Y.: A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. Circ. Res. 68(6), 1501–1526 (1991)
3. Sato, D., Xie, Y., Weiss, J.N., Qu, Z., Garfinkel, A., Sanderson, A.R.: Acceleration of cardiac tissue simulation with graphic processing units. Med. Biol. Eng. Comput. 47, 1011–1015 (2009)
4. Plank, G., Liebmann, M., Santos, R.W., Vigmond, E.J., Haase, G.: Algebraic multigrid preconditioner for the cardiac bidomain model. IEEE Trans. Biomed. Eng. 54(4), 585–596 (2007)
5. Sundnes, J., Lines, G.T., Cai, X., Nielsen, B.F., Mardal, K.A., Tveito, A.: Computing the Electrical Activity in the Heart. Springer (2006)
6. Maclachlan, M.C., Sundnes, J., Spiteri, R.J.: A comparison of non-standard solvers for odes describing cellular reactions in the heart. Comput. Methods Biomech. Biomed. Engin. 10, 317–326 (2007)
7. Bell, N., Garland, M.: Efficient Sparse Matrix-Vector Multiplication on CUDA. Tech. rep., NVidia Corporation (2008)
8. Santos, R.W., Plank, G., Bauer, S., Vigmond, E.J.: Parallel multigrid preconditioner for the cardiac bidomain model. IEEE Trans. Biomed. Eng. 51(11), 1960–1968 (2004)
9. Rocha, B.M., Campos, F.O., Amorim, R.M., Plank, G., dos Santos, R.W., Liebmann, M., Haase, G.: Accelerating cardiac excitation spread simulations using graphics processing units. Concurrency and Computation: Practice and Experience (2010)
10. Weber, R., Gothandaraman, A., Hinde, R.J., Peterson, G.D.: Comparing Hardware Accelerators in Scientific Applications: A Case Study. IEEE Transactions on Parallel and Distributed Systems 22, 58–68 (2011)
11. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing Company (1996)

Fine Grained Parallelism in Recursive Function Calls

Dimitris Saougkos, Aristeidis Mastoras, and George Manis

University of Ioannina, Dept. of Computer Science
P.O. Box 1186, Ioannina 45110, Greece
manis@cs.uoi.gr
http://cs.uoi.gr/~manis

Abstract. Most recursive functions hide a significant degree of inherent parallelism. Much research work has been done in detecting and exploiting this parallelism, mainly by focusing on calls which can be invoked in parallel. However, not all recursive algorithms allow concurrent execution of function calls. In this paper we study how we can extract concurrency from those recursive functions which cannot be parallelized with the commonly used methods. The key idea is to parallelize them in a finer level, using as infrastructure a multi-core, multi-thread architecture. Multi-core architectures give us the opportunity to achieve higher degree of fine grained parallelism than the conventional parallel architectures, by providing real thread level parallelization and low overhead in the communication between different threads. For our experiments we use the SVP processor and model, a novel multi-core architecture which supports threads of execution with rapid communication between "neighboring" threads. The mapping of recursive functions onto the SVP processor is part of a more general framework the C2μTC/SL source to source compiler, developed to support the automatic extraction of parallelism from C programs and the exploitation of the special characteristics of the SVP processor. The experimental results are very encouraging and show satisfactory speedups between the parallel execution and the automatically produced code.

Keywords: recursive function calls, automatic parallelization, SVP, self-adaptive virtual processor.

1 Introduction

It is common practice for the programmers to write recursive programs without taking into account the overhead introduced by the successive function calls or from possible repetition of computations. Compilers often use techniques like tail recursion in order to speed up the execution of recursive functions. Parallelization techniques have also been appeared in which the compiler tries to extract concurrency from the recursive function calls. Much work has been done in this field since recursive functions usually hide a high degree of parallelism.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 121–130, 2012.

In the very popular divide and conquer technique some input is divided into two or more parts and a recursive call is invoked for each one of those parts. Usually each call operates on a different part of the data and, thus, it is safe to perform these recursive calls in parallel. For example, let us consider the quicksort algorithm. The array is divided into two smaller arrays, and one recursive call is performed for each one of the smaller arrays. Since each call performs on different data it is safe to invoke both calls in parallel. This is an easy task for a parallel programs developer since it can be easily understood that each call performs on different data. The problem becomes difficult for a compiler, where the detection of parallelism must be automatic. A framework for automatic parallelization of such recursive functions is discussed in [4] where the main target is the divide and conquer algorithms. At compile time symbolic array section analysis is used to detect the interdependence of multiple recursive calls in a procedure. When the compile time analysis is not enough, speculative run time parallelization is employed. Similar work is described in [9] where parallelism from recursive functions targeting divide and conquer algorithms is also extracted. In [3] the Huckleberry tool is presented, again detecting parallelism from divide and conquer algorithms and producing code for a multi-core platform. A quantifier-elimination based method belongs in the same family too. This method shrinks function closures representing partial computations, splits the input structure and perform computation on each part in parallel [8].

However, when a recursive call does not call more than one instance of itself, the extraction of parallelization is more difficult. Let us consider the factorial as an example. The computation of `fact(n)` requires that the computation of `fact(n-1)` has been completed. Thus, the kind of parallelism discussed above cannot be detected here. We target this kind of recursive function on which the commonly used methods cannot be applied and try to exploit threads of execution in order to achieve a finer lever of parallelism in the instruction level. For our experiments we use the SVP processor and model and exploit the hardware level threads of execution and the synchronizing memory which allows neighboring threads to communicate with low overhead. This work is part of the C2μTC/SL compiler, a source to source parallelizing compiler which automatically maps C programs onto the SVP model.

In similar work, we must also mention recursion splitting [5] a technique which converts recursive functions into loops and then applies loop parallelization techniques on them. In [1] an analytical method is presented which transforms recursive functions on general recursive data structures into compositions of parallel skeletons. Both papers use functional languages.

The rest of the paper is structured as follows. Section 2 gives an abstract presentation of the SVP model and briefly presents the C2μTC/SL compiler. Section 3 discusses how we map the recursive functions onto the SVP processor while in section 4 the experimental results are presented. The last section presents plans for future extensions and summarizes this work.

2 The Processing Model and the C Parallelizing Compiler

In SVP [2,6], the unit of parallelism is a thread. Threads are organized in groups which are called *families*. The threads in a family are ordered, i.e. each thread has a unique number which defined the order of the thread in the family. The communication between threads can be done through (i) the shared memory and through (ii) the synchronizing memory. The shared memory can be considered as consistent only after all threads in the family has terminated. The synchronizing memory allows data to be forwarded from thread with index i to the thread with index $i+1$. No other kind of communication is possible between any two executing threads. Figure 1 present schematically the processing model.

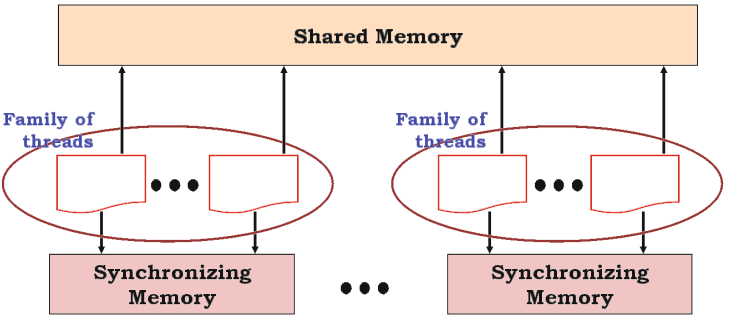


Fig. 1. The processing model

To better describe the SVP model, an intermediate language has been defined. It is called *micro-threaded C* (μTC) [7] and it is similar to other concurrent languages based on C. SL is a script language on top of μTC which masks some details and provide a more user friendly interface to the programmer.

C2μTC/SL is a parallelizing compiler which maps sequential C programs on the SVP concurrent processing model and the μTC and SL languages. The aim of C2μTC/SL is to extract parallelism from C programs and produce families of threads in order to exploit the special characteristics of SVP. Since we expect to extract most of the parallelism of a program from loop structures C2μTC/SL mainly focuses on loops [10]. Contrary to most parallelizing compilers, C2μTC/SL moves the problem of scheduling from compile-time to run-time [11]: at compile-time a lightweight scheduler is generated which in runtime will be responsible for the coordination of the execution of the families allowing more flexibility than static scheduling.

We will give an example to show how we can achieve parallelism in SVP. Let us consider the inner product:

```
sum=0;
for (i=0;i<N;i++)
    sum+=a[i]*b[i];
```

The compiler will produce code which will create one family consisted of N threads of execution. Each thread i will perform the multiplication `a[i]*b[i]` and then wait to receive the partial sum for the thread i-1, namely it will receive the amount $sum = \sum_{j=0}^{i-1} a[j]b[j]$. After receiving the partial sum from the previous thread, it will add its own contribution and forward the result to thread $i+1$, namely it will forward the amount $sum = \sum_{j=0}^{i} a[j]b[j]$. All multiplications `a[i]*b[i]` will be performed in parallel and only the summation of the multiplications will be serial. For the summation procedure the synchronizing memory will be used which allows fast forward of data from thread i to thread $i+1$. The execution times for the serial execution and the execution on SVP is shown on figure 2.

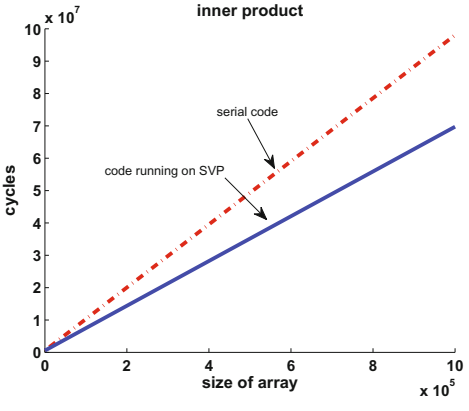


Fig. 2. Execution times for the inner product

3 Recursive Functions onto SVP

A recursive function call consists of successive calls of the same function each one of which performs on different data. These data have been produced by previous calls of the function in the chain of the function calls, or are common for all calls. In most cases each function uses the data produced by its calling functions, or at least uses data transferred through one of its calling functions.

When mapping a recursive function onto SVP, a family of threads is created and each one of the successive recursive calls is assigned to one thread of execution. Threads in a family are ordered. Each thread i in the family can communicate with the thread $i+1$ and forward data in a way similar to the "return" call of the recursive function. Obviously, the first thread in the family corresponds to the last function call in the chain of the recursive functions, the second thread to the one before the last, etc.

The information needed to be extracted is how many threads are to be created and what kind of communication is required between them. Of course, the great challenge is to do this automatically, without any hint from the programmers.

In case we fail to extract this information it is better not to attempt to parallelize the execution at all and choose to run the program sequentially.

The structure of a recursive function call is not the same for all problems. However we can observe that in most cases this structure is of the form:

```
if (...) return(...)
else if (...) return(...)
else if (...) return(...)
...
else return(...)
```

When the structure of the function is as shown above, it is possible to decide how many threads have to be created and what kind of communication is necessary between them. The steps we have to follow are:

1. we decide which variable can be used as an index. In most recursive functions such a variable exists and can be located in the conditions of the `if` statements and in the parameters of the function
2. we detect for which values of this variable it is not necessary to call the function recursively
3. we detect how the value of this variable changes for any two successive calls in the chain of the recursive calls
4. based on steps 2 and 3, we identify how many recursive calls will be done. We create a family with the same number of threads
5. we decide on what data should move from one thread to the next thread in the family
6. we decide on what data are global to all threads, i.e. are not modified in the body of the function call
7. we detect the initial values for the variables of the first threads in the family
8. we compile the recursive function and produce the SL code

Let us consider a small example here, the computation of the factorial:

```
long int factorial(int x)
{
    if (x<0) return(-1); // not defined
    else if (x>L) return(-2); //overflow
    else if (x==0) return(1);
    else return(x*factorial(x-1));
}
```

According to the steps discussed above, the following information must be extracted.

1. the variable which can be considered as the *index* is x. The variable x exists in all conditions and in the parameters of the function
2. the function will not perform any recursive calls when $x < 0$ or $x > L$

3. each recursive call is based on data returned by the next recursive call. The value of x is reduced by 1 in every call, according to the x–1 actual parameter in the call
4. suppose we want to compute the factorial of N. Based on steps 2 and 3 we can conclude at compile time that we need N recursive calls. Thus, we must create a family with N threads
5. each thread forwards to the next thread in the family an integer, which is equal to the return value of the corresponding function call
6. the first thread is initialized to 1, the return value of the function for $x = 0$. This value is found if we move downwards (as indicated in step 3) from the given value until we found a value the computation of which does not require a recursive call
7. the information extracted in the steps above is enough to automatically produce a program in SL which produces the same results with the function `factorial` and can execute in parallel exploiting the special characteristics of the SVP architecture.

The expected speedup of mapping such a recursive call onto the SVP computing model can be summarized in the following steps:

1. the execution environment for each thread is created in parallel and not sequentially as in recursive calls
2. the part of the computation which performs the evaluation of the condition is executed in parallel, since the value of x is given for each thread from the beginning (x is the index of the thread inside the family)
3. the rest of the computation, before the recursive call, can also run in parallel. In this example, x can move to the register and wait for the second operand before the multiplication can be performed. Please note that this part of the computation can be more expensive or much expensive. For example, if we had x^3 or $\sqrt{x}$ or generally $f(x)$ instead of x the computation before the recursive calls becomes more expensive.

Let us now discuss another one interesting example, the fibonacci numbers:

```
long int fib(int x)
{
    if (x<0) return(-1); // not defined
    else if (x>L) return(-2); //overflow
    else if (x==0) return(1);
    else if (x==1) return(1);
    else return(fib(x-1)+fib(x-2));
}
```

In this example the function `fib` is called recursively twice. We can again exploit the synchronizing memory and employ two shared variables this time, one for keeping the fibonacci number computed by the thread $i-1$ and one for the fibonacci number computed by the thread i–2. In this way we eliminate the need for each function call to call itself twice. This leads to a significant reduction of the necessary number of threads and to a further improve of the execution time, even more significantly this time as we will see in the following section.

4 Experimental Results

We will present two experiments with well known problems. Experimental results have been collected using a hardware simulator of the SVP model [6]. The simulator counts machine-cycles necessary for a program to complete. Both serial and parallel code run on the simulator. The serial version is pure C, i.e. we do not use any the special characteristics of SVP (synchronizing memory, families, e.t.c.).

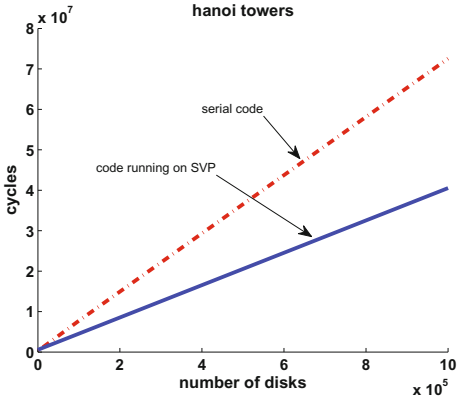
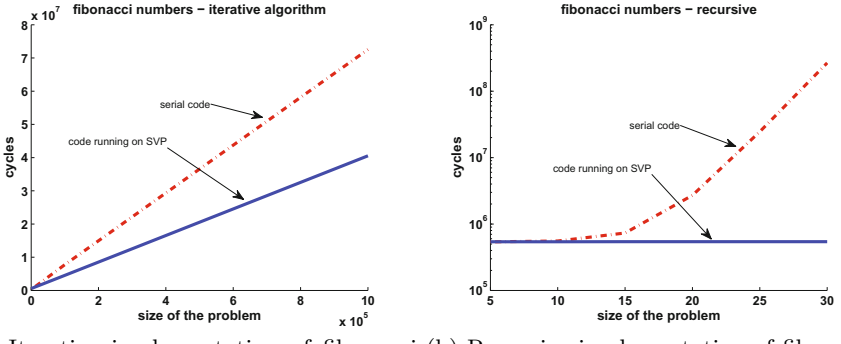


Fig. 3. Execution times for Hanoi towers



(a) Iterative implementation of fibonacci numbers.

(b) Recursive implementation of fibonacci numbers. Y-axis is in logarithmic scale.

Fig. 4. Execution times for Fibonacci numbers

In the first experiment we compute the numbers of the moves necessary to solve the Hanoi tower problem with N disks. The recursive code that solves the problem follows:

```
long int hanoi(int x)
{
    if (x<0) return(-1); // not defined
    else if (x==0) return(0);
    else return(2*hanoi(x-1)+1);
}
```

In order to compute `hanoi(x)` the function `hanoi(x-1)` has to have finished the computation. The problem is serial, at least in its current form. A family of N threads will be created, all initializations will take place in parallel, all evaluation of conditions will also be executed in parallel and only the final computation in every thread will be serial. Thus, we expect an acceleration of the execution due to this parallelism. Execution times for the hanoi problem and for different number of disks is shown on figure 3.

Next we considered the fibonacci numbers problem. The code is listed in the previous section. Even though `fib(x-1)` and `fib(x-2)` can be invoked in parallel, there is replication of work which results in low execution times, even if we run it in parallel. We avoid the replication of work by moving two amounts from thread to thread, shifting the value of the first to the second and computing only the first amount. This leads to a significant speedup even if compare with the iterative implementation of fibonacci numbers. The execution times for the iterative implementation is shown on figure 4(a) and the execution times for the recursive implementation is shown on figure 4(b). Please note that the recursive implementation can compute only the first few fibonacci numbers, and even in such a small problem size the execution time explodes. The reason that the line for SVP in figure 4(b) seems flat is that the y-axis is in logarithmic scale.

Please also note that the recursive hanoi and the iterative fibonacci numbers presents similar execution times. This is due to the optimizations performed by the gcc compiler which eliminates the recursion. The execution times on SVP is still better. In the fibonacci problem (i) gcc fails to perform optimizations and (ii) there is much replication of work, something that explains why the acceleration is that large.

We presented experimental resutls comparing the execution times of the code automatically produced by the compiler and the sequential code. We also wrote parallel code manually. The experimental resutls were similar to that from the code produced by the compiler.

5 Discussion

Research work on automatic extraction of parallelism from recursive function mainly focuses on those recursive calls for which the data dependences allow more than one function call to execute in parallel. For example, the quicksort separates an array into two subarrays and perform one function call on each one of the arrays independently. We examine a different model in which all function calls start execution and there is a synchronization point between successive function calls in which one function call waits for data from the previous one.

Such recursive function calls is the factorial, the computation of the power and more general all recursive function calls that can be transformed into loop structures. Even though the execution of these calls seems to be necessarily sequential, we exploit the SVP and the special purpose hardware to achieve parallelism and, thus, acceleration of execution. The parallelism extracted is due to divide and conquer technique and cannot be compared with that.

Comparison with similar work is meaningful when considering the ability of most other compilers to transform recursive calls into loops. Our compiler extracts the same information from loops with those compilers; however, we map this information on a parallel architecture/model and not on a sequential loop structure, accelerating the execution of an application that seems completely sequential. The most important contribution of this paper is that we showed that this is possible to do it and examined how. The experimental results verified our claim.

The compiler at the present stage of design and development can extract parallelism from most well-written function calls. Generally it takes the conservative approach and transforms only programs for which it is absolutely sure that the extracted information is correct. Generally the extraction of the necessary information is difficult and sometimes even undecidable. However, for most of the well written code this information is possible to be extracted at in most of the cases not difficult. Our purpose was not to invest in capturing the widest range of written code, especially the not the tricky one. Our purpose was to show that the automatic capturing of parallelism is feasible and the acceleration of the execution possible. I would characterize our analysis as "simple analysis with a big number of classical cases". Functions from which parallelism cannot be extracted from the algorithm described in this paper remain without transformation.

6 Conclusions and Future Extensions

The parallelization method presented above is confined to recursive functions which use integer variables updated in a regular and systematic way. A large number of recursive functions uses integer variables regularly updated between successive calls. However not all recursive functions belong in this category. This method can be generalized to include some function calls from linked lists as well. Linked lists have also a regular structure while moving from one node to another becomes with a systematic way, well defined and easily extracted way. Similar to the approach presented in this paper, a number of threads equal to the number of nodes of the list can be created but using a different mechanism this time. Then, each thread can run the code up to the point where the function is called recursively. The synchronizing memory will be used for moving data between successive calls.

In this paper we presented a part of the C2μTC/SL source to source compiler which maps recursive functions onto the SVP processing model. The compiler extracts information from the source code and produces a family with the required number of threads, one for each expected recursive call. Experimental results show that we can obtain remarkable speedup, even when the gcc compiler successfully eliminates the recursion.

References

1. Ahn, J., Han, T.: An analytical method for parallelization of recursive functions. Parallel Processing Letters, 87–98 (2000)
2. Bernard, T., Bousias, K., Guang, L., Jesshope, C., Lankamp, M., van Tol, M., Zhang, L.: A general model of concurrency and its implementation as many-core dynamic RISC processors. In: Proc. of SAMOS 2008, Samos, Greece (2008)
3. Collins, R.L., Vellore, B., Carloni, L.P.: Recursion-driven parallel code generation for multi-core platforms. In: Design, Automation and Test in Europe, DATE (2010)
4. Gupta, M., Mukhopadhyay, S., Sinha, N.: Automatic parallelization of recursive procedures. Int. J. Parallel Program. 28(6), 537–562 (2000)
5. Harrison, W.L.: The interprocedural analysis and automatic parallelization of Scheme programs. LISP and Symbolic Computation 1(1), 35–47 (1990)
6. Jesshope, C., Lankamp, M., Zhang, L.: The implementation of an SVP many-core processor and the evaluation of its memory architecture. SIGARCH Comput. Archit. News 37, 38–45 (2009)
7. Jesshope, C.R.: μTC - an intermediate language for programming chip multiprocessors. In: Proc. of Pacific Computer Systems Architecture Conference 2006, Shanghai, China (2006)
8. Morihata, A., Matsuzaki, K.: Automatic Parallelization of Recursive Functions Using Quantifier Elimination. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 321–336. Springer, Heidelberg (2010)
9. Rugina, R., Rinard, M.: Automatic parallelization of divide and conquer algorithms. In: PPoPP 1999: Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 72–83 (1999)
10. Saougkos, D., Manis, G.: Specifying loop transformations for C2μTC source to source compiler. In: Proc. of Compilers for Parallel Computers, Zurich, Switzerland (2009)
11. Saougkos, D., Manis, G.: Run–time scheduling with the C2μTC/SL parallelizing compiler. In: Proc. of 2nd Workshop on Parallel Programming and Run–Time Management Techniques for Many-core Architectures, Como, Italy (2011)

On-Line Grid Monitoring Based on Distributed Query Processing

Bartosz Balis[1], Grzegorz Dyk[2], and Marian Bubak[1,2]

[1] AGH University, Department of Computer Science, Krakow, Poland
[2] AGH University, ACC Cyfronet AGH, Krakow, Poland
{balis,bubak}@agh.edu.pl

Abstract. Existing Grid monitoring approaches do not combine three desirable features: on-line access to monitoring data, advanced query capabilities and data reduction. We present a solution for on-line monitoring of large-scale computing infrastructures based on Complex Event Processing principles and technologies. We focus on leveraging CEP for distributed processing of client queries and monitoring data streams. This results in significant reduction of network traffic due to on-line monitoring. We discuss benefits of CEP-based approach to monitoring and describe details of processing queries in a distributed way. A case study – monitoring of load caused by jobs in a Grid infrastructure – is presented. Performance evaluation to investigate monitoring overhead in terms of CPU, memory and network traffic is also provided.

Keywords: on-line monitoring, grid infrastructure, complex event processing.

1 Introduction and Motivation

On-line monitoring in large-scale computing infrastructures is crucial for such purposes as SLA contract monitoring [5], system intrusion detection [9], self-healing [6] or performance steering and dynamic reconfiguration [11]. Existing Grid information systems store information as data sets and do not expose direct streams of on-line measurements [1,8]. Some monitoring approaches provide on-line access to monitoring data streams [7,4,10], but they fail to provide advanced query capabilities and sufficient data reduction.

We present an on-line distributed monitoring solution for large-scale computing infrastructures based on Complex Event Processing. The basic concept of CEP-based monitoring and its implementation – the GEMINI2 system – have been presented in [2]. In this paper we focus on distributed processing of queries and monitoring data streams in order to minimize resource consumption due to monitoring. Such an approach allows for the combination of three desirable features in a Grid monitoring system: (1) on-line access to monitoring information, (2) advanced query capabilities that enable on-demand definition and real-time calculation of complex derived metrics, (3) data reduction which prevents flooding of network with monitoring data.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 131–140, 2012.

This paper is organized as follows. Section 2 presents related work. Section 3 introduces the concept of CEP-based on-line monitoring and the GEMINI2 architecture. In section 4, distributed processing of requests and monitoring event streams is explained. Section 5 presents the resource load monitoring use case. In section 6, monitoring overhead is investigated. Section 7 concludes the paper.

2 Related Work

While there are many solutions for Grid monitoring, below we overview these which provide on-line monitoring combined with query capabilities. MonALISA [7] provides two mechanisms to request on-line streams of monitoring data: predicates and data filters. However, predicates only allow to request data matching a specific regular expression. Data filters potentially enable to calculate any derived values from collected ones (aggregations or even distributed correlations). However, they need to be developed as Java plugins. CEP, on the other hand, offers a generic mechanism whereby derived values can be defined on-demand and calculated in real-time simply by using a *continuous query language*.

In vizTool [3], events from monitoring of a distributed system are gathered on-line and visualized. However, query capabilities are limited to defining of custom filters. In addition, a limited number of high-level metrics (such as aggregation) can be computed at the level of visualization.

In R-GMA [4], monitoring data is collected by distributed producers and stored in local relational databases. A global registry of producers is also provided making it possible to formulate distributed, continuous queries using SQL as the query language. However, the SQL and the relational model have not been designed for continuous querying over streams. Therefore, this approach is inherently restricted in comparison to CEP, lacking such key capabilities as aggregations over sliding window, stream joining or event correlation. In addition, querying distributed relational databases is arguably less efficient than in-memory processing of data streams in a CEP engine.

SCALEA-G [10] also offers on-line monitoring, albeit with limited query capabilities. A client essentially can choose which entities to monitor, select desired metrics, and optionally specify XQuery or XPath filters (data is represented in XML). Besides lacking the possibility of defining derived metrics and distributed queries, all remarks about the SQL/relational model are also valid for XQuery/XML which is not designed for data streams.

In summary, existing Grid monitoring systems which offer on-line monitoring, lack advanced query capabilities and/or do not support distributed processing.

3 Benefits of Applying CEP to On-Line Monitoring

CEP offers several benefits when applied to on-line monitoring, notably real-time access to monitoring data, on-demand definition and calculation of derived metrics, and data reduction.

Real-Time Access to Monitoring Data. In the CEP-based approach, monitoring data is encapsulated as events (named collections of attributes) which typically contain information about the *monitored resource* and associated *metrics*. An example *HostInfo* event could contain a unique host name, current CPU load, current memory consumption, etc. Next, sensors need to be deployed which generate the events and feed them to a CEP engine. The clients retrieve monitoring information from the engine using queries expressed in a continuous query language. For example, in the simple query `select * from HostInfo(name='zeus.cyfronet.pl')`, attribute selection and filtering is used in order to subscribe to the event stream of all metrics for the specified host.

On-Demand Definition and Calculation of Derived Metrics. Decision making based on on-line monitoring typically requires real-time calculation of derived metrics on the basis of data from multiple sources collected over time. Such derived metrics can be defined by clients *on-demand* through complex queries using such constructs as aggregations over sliding window, stream joining, event correlation or results grouping and ordering. Example metrics not difficult to express in a continuous query language are as follows. (1) *Return all host names where average CPU consumption over last 5 minutes exceeds 90% AND the pid of the process which has the highest CPU consumption on the given host.* (Aggregation over sliding time window, conditional output, joining streams). (2) *Return alert when an average response time of client requests exceeds 100ms OR average data transfer rate drops below 128KB/s.* (Event pattern).

Data Reduction. Data reduction aims at minimizing network traffic due to on-line monitoring. This can be achieved with a distributed processing of monitoring requests, as explained in the next section.

4 Distributed Request Evaluation

4.1 Distributed Architecture and On-Line Processing

Fig. 1 presents a distributed architecture of the GEMINI2 framework. The architecture features a hierarchy of distributed Monitors and sensors, all of which may contain CEP engines. The Monitors are organized into a super-peer architecture and share information regarding available sensors. A client request, submitted to any Monitor, is analyzed and distributed to other affected Monitors and, ultimately, sensors. The distributed processing involves two stages: (1) distribution of client requests, (2) assembly of partial monitoring data streams. In the distribution stage, a client sends a query expressed in the Event Processing Language to a Monitor. The query is analyzed and decomposed into partial statements which are distributed to affected Monitors and sensors. In the assembly stage, the partial monitoring data streams are assembled in order to produce a final result matching the original query.

The task of request decomposition is carried out by a dedicated *distributor* component. Its function is to analyze a given EPL statement and calculate (1)

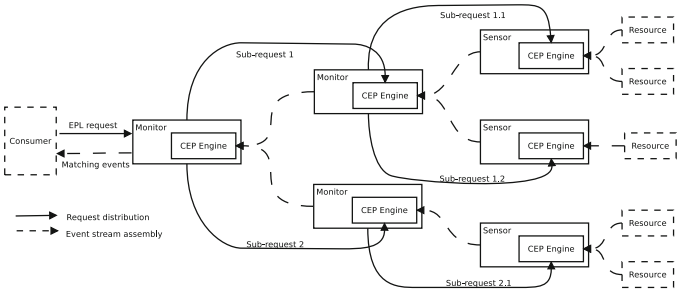


Fig. 1. Distributed architecture of the GEMINI2 monitoring infrastructure. CEP engines deployed in Monitors and (optionally) sensors enable the distribution of queries and assembly of partial replies.

one or more *partial statements* for individual producers, (2) the *assembly statement*. The partial statements are sent as EPL requests to appropriate lower-level producers. The *assembly statement* is installed in the CEP engine of the distributor in order to process the partial event streams from producers and assemble them into a final result expected by the consumer. Consider the following example EPL statement:

```
select hostName, avg(cpu.load) as cpuLoad
        from CpuInfo.win:time(10 minutes) as cpu
        group by hostName output last every 5 minutes
```

This request returns, once every 5 minutes, events containing the average CPU load over last 10 minutes for each monitored host. The *group by* clause results in generating a separate event for each unique host. This request is decomposed into the following partial statements:

```
select hostName, sum(cpu.load) as loadSum, count(cpu.load) as loadCount
        from CpuInfo.win:time(10 minutes)
    group by hostName output last every 3 minutes
```

The partial streams generated by these statements are assembled using the following assembly statement:

```
select hostName, sum(loadSum)/sum(loadCount) as cpuLoad
        from assemblyStream.std:unique(producerId, hostName)
    group by hostName output last every 5 minutes
```

4.2 Distribution Patterns

There is a number of continuous query language constructs whose influence on the effectiveness of distribution should be investigated. These constructs include *aggregations* (calculation of min, max, average etc.), *sliding windows* (time- and length-based), *output control statements* (grouping, ordering etc.), and *stream joining* (selecting attributes from multiple streams joined by a common attribute value). Some distribution patterns are presented in Table 1.

The distribution of the `max` aggregate function utilizes a `unique` window. It works as a buffer of length 1 for events with same value of a given attribute (in this case `producerSpec`). Therefore, only the latest event from each producer

will be used to compute the final result. The distribution of a request containing a length-based sliding window requires dividing the size of the partial length windows by the number of producers so that the total number of events coming to the consumer matches the original request.

Table 1. EPL distribution pattern examples. The rightmost column contains, in order, the partial statement(s) and the assembly statement. The notation used in the statements is as follows:
– `assemblyStream` denotes an event stream composed of events coming from producers that received partial statements.
– `producerId` is a unique identifier assigned to event producers.
– `producerCount` denotes the number of producers that received partial statements.
– `[generic expression]` indicates a part of the request whose precise content does not influence the distribution (e.g., `[aggregate]` denotes any aggregate function).

Request type	Distribution / Assembly

Calculation of `max` aggregation function
 	
`select max(value) as [alias] from [stream];`
	`select max(value) as mx from [stream];`
	
	`select max(mx) as [alias] from assemblyStream.std:unique(producerSpec);`

Aggregation with sliding time window: `select [aggregate] as [alias] from [stream].win:time([time spec])`	(same as partial for normal aggregate but over time window with [time spec]) (same as assembly for given aggregate)
Aggregation with sliding length window: `select [aggregate] as [alias] from [stream].win:length( [length spec])`	(same as partial for normal aggregate but over length window with length [length spec]/producerCount) (same as assembly statement for given aggregate function)

The distribution of statements involving *joining of multiple streams* (example given in Section 5) is particularly challenging. Two basic cases can be identified. (1) All events in all streams involved in a join are provided by the same producer; in this case the join operation is resolved entirely within this particular producer and the distribution is trivial (the same as if there was no join). (2) At least one of the event streams involved in the join is provided by a different producer than the rest; the distribution is impossible as there must be a single CEP engine processing all streams.

4.3 Semantics of Metrics in Distributed Evaluation

The value of derived metrics expressed in EPL depends on temporal relationships between events. Consequently, values calculated in a distributed way may differ from those obtained in a centralized way. This is caused by processing in multiple distributed engines and associated delays it involves. The more layers in the distributed hierarchy, the more visible this effect can be.

The problem is well illustrated in requests with time- and length-based sliding windows, as depicted in Fig. 2. The y axis represents time. The arrivals of three events are shown for two different settings (dashed events represent the centralized case, solid lines – distributed one, delays being significantly higher). The events are inserted into a time-based sliding window. Because of the delays, the contents

of the sliding windows vary in the different settings. As a result, calculations such as aggregate functions operating on those windows will return different results in each case. A similar effect may occur for length-based windows: depending on the sequence of event arrivals, different events will fit into a given window.

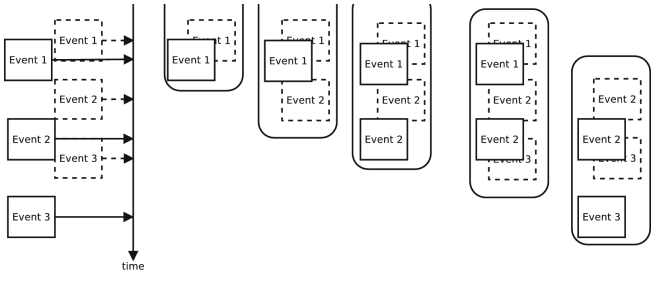


Fig. 2. Effect of different event arrival delays on time-based sliding window calculations. Dashed events represent centralized evaluation, solid – distributed one. The arrangements of events in windows vary in both cases.

Given the described examples, we can say that the request semantics varies slightly depending on the setting – distributed or centralized. The question is whether the calculations obtained in the distributed semantics are actually *incorrect* or not? The answer depends on the way the calculations are used. In on-line monitoring it is clear these 'inconsistencies' are not significant. First, the difference is the greater the smaller the time window is. However, decisions based on on-line monitoring are rarely made on the basis of short-term (instant) readings. Rather, a longer trend is taken into account. The trend, however, is clearly preserved regardless of the incidental temporal fluctuations.

5 Case Study: Resource Load Monitoring

Let us consider the following use case to demonstrate on-line CEP-based monitoring and a distribution of a complex EPL statement. Suppose a site administrator would like to receive continuous reports about machines with the highest load in a cluster and top processes whose CPU consumption is the highest.

An example deployment of computing resources and GEMINI2 monitoring components is shown in Fig. 3. The environment consists of two sites (SARAH and DIANA) in which there are clusters of Worker Nodes managed by Computing Elements. GEMINI2 sensors are deployed on Computing Elements and Worker Nodes. The latter generate two types of event streams: (1) *HostInfo* – one stream for each host, contains, among others, the host's name and dynamic metrics: *cpuLoad*, *memFree*, etc., (2) *ProcInfo* – one stream for each process, contains the process' pid, the name of the host on which the process is running, and metrics: *cpuLoad*, *memUsage*, etc. GEMINI2 Monitor is deployed on a global level (Front Monitor) on top of multiple Grid sites; intermediate Monitors are also deployed at each site.

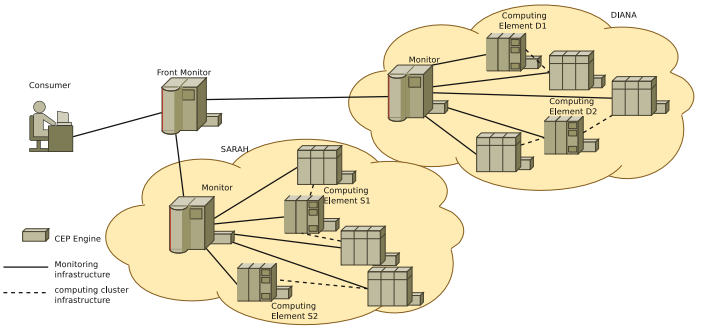


Fig. 3. Example Grid computing environment with GEMINI2 infrastructure components deployed

The administrator requests to be notified about all Worker Nodes in the cluster DIANA whose average CPU load over last 5 minutes exceeded 90%. In addition, a list of top 10 processes on those hosts in terms of CPU consumption should be returned. This can be expressed in EPL in the following way:

```
select host.name, avg(host.cpuLoad), proc.pid, avg(proc.cpuUsage)
    from HostMs(CE='DIANA').win:time(5 min) as host,
         ProcessMs(CE='DIANA').win:time(30sec) as proc
    where proc.hostName = host.name  /* join streams */
    group by host.name, pid /* group by unique (hostname,pid) */
    having avg( host.cpuLoad ) > 90
    output all every 1 minute
    order by avg(proc.cpuUsage) desc
    limit 10                        /* display top 10 results */
```

The Front Monitor determines that the only producer for this request is the Monitor on site DIANA, therefore the statement is simply forwarded to this monitor, while the Front Monitor installs in its CEP engine the following assembly statement: `select * from aggregateStream`.

The Monitor in the DIANA cluster determines that all sensors on Worker Nodes are affected by the query. The statement is decomposed into the following partial statements submitted to sensors:

```
select host.hostName, sum(host.cpuLoad) as hostCpuLoadSum, count(host.cpuLoad
    ) as cpuLoadCount, proc.pid as pid, sum(proc.cpuUsage) as procCpuUsageSum
    , count(proc.cpuUsage) as procCpuUsageCount
    from HostMs(CE='DIANA').win:time(5 min) as host,
         ProcessMs(CE='DIANA').win:time(30 sec) as proc
    where proc.hostName = host.name
    group by host.name, proc.pid
```

The assembly statement is as follows (*producerId* being the name of a Worker Node):

```
select hostName, sum(hostCpuLoadSum)/sum(hostCpuLoadCount) as hostCpuLoad,
    pid, sum(procCpuUsageSum)/sum(procCpuUsageCount) as procCpuUsage
    from assemblyStream.std:unique(producerId, hostName, pid)
    group by hostName, pid
    having sum(hostCpuLoadSum)/sum(hostCpuLoadCount) > 90
    output all every 1 minute
    order by sum(procCpuUsageSum)/sum(procCpuUsageCount) desc
    limit 10
```

6 Performance Evaluation

In this section, we measure the monitoring perturbation of the employed solution for on-line monitoring in terms of CPU utilization, memory consumption and network traffic.

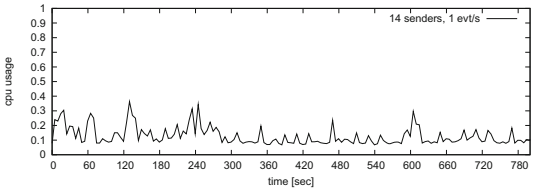


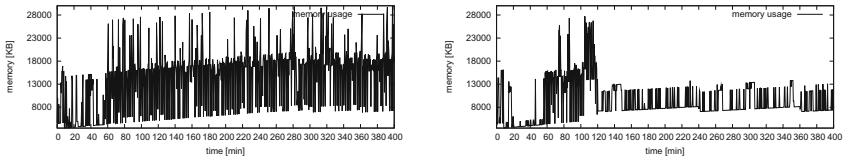
Fig. 4. CPU consumption for Esper engine having 1000 EPL statements registered of following format: `select avg(userTime) from CpuInfoEvent(machine='{host}').win:time(60 seconds) output last every 30 seconds` where {host} is different for each statement

CPU Utilization. Low CPU utilization is critical on the sensor side where also jobs run. In GEMINI2, the CPU utilization depends on the intensity of event arrivals to the CEP engine[1]. Because CEP engines deployed in sensors are not exposed to high event rates (given the fact they only collect events from a single node), the associated CPU usage is very low – it almost never exceeds 1%. The Monitors, on the other hand, collect events from many sensors therefore the resulting CPU usage can be high. This does not pose a problem though, since Monitors can be deployed on dedicated servers. Fig. 4 presents a benchmark of CPU usage caused by a Monitor in the presence of 14 event senders (sensors), each producing 1 event per second, and 1000 complex queries registered in the CEP engine. The Monitor and the senders were working on a double-core 2.2-GHz-per-core machine. Even for such high parameters, the resulting CPU usage is moderate; most of the time it balances at about 10%, almost never exceeding 30%.

Memory Utilization. Memory utilization due to CEP-based processing is caused mainly by large sliding windows. However, CEP engines are optimized also in this respect. Fig. 5(a) presents memory usage caused by a GEMINI2 sensor for an EPL statement with a four-hour long sliding time window. The event frequency was 1 per second and output was released every hour. Memory utilization slowly increases for the first 4 hours (filling up the window) and then remains constant.

Figure 5(b) presents results for a smaller, two-hour window. In addition, all events are released every 2 hours. This causes memory usage to decrease periodically which is clearly visible in the graph. Overall, given the amount of memory on modern computing nodes (Gigabytes), the memory overhead is not significant.

[1] Compare performance benchmarks for the Esper engine `http://esper.codehaus-.org/esper/performance/performance.html`, Accessed 10.05.2011.



(a) `select avg(userTime) as au from CpuInfo.win:time(4 hours) output last every 1 hour`

(b) `select * from CpuInfo.win:time(2 hours) output all every 2 hours`

Fig. 5. Memory utilization by sensors for two different EPL statements with long sliding windows and event generation rate 1 per second

Network Utilization. Distributed processing can significantly reduce network bandwidth utilization due to on-line monitoring. In order to investigate this, we measured the network traffic in loopback interface between artificial sensors with CEP engine and a Monitor working on same machine. The system was processing statement `select avg(usertime) as au from CpuInfo.win:time(10 seconds) output last every 10 seconds`. The measurements have been made for both centralized and distributed configurations. In the first case no statement distribution was performed and the sensors were continuously sending all available monitoring data every second. In the second measurement the statement was decomposed and distributed to sensors. As a result, partial aggregation was performed in sensors and sent to the Monitor every 10 seconds.

While the employed request is simple, it is designed to generate a relatively high network traffic when no data reduction is employed. Enabling the distribution allowed to *reduce the network traffic 5-fold*. Given that the trade-offs in additional CPU and memory consumption are minimal, this is a promising result.

7 Conclusion

We have presented an approach to on-line monitoring of large-scale computing infrastructures based on Complex Event Processing. CEP, besides providing the capability for on-line access to monitoring data streams, offers the following benefits: (1) Advanced derived metrics can be defined on-demand simply by using a continuous query language (such as EPL), without the need to develop dedicated plug-ins, or even to reconfigure the monitoring system. (2) Query capabilities offered by the continuous query languages are more advanced than in alternative approaches (SQL, XML/XQuery). (3) The derived metrics are calculated in real-time by dedicated CEP engines, optimized for performance and small footprint. (4) Monitoring overhead in terms of network traffic can be minimized by applying distributed processing and sensor-side CEP engines. Thanks to the small footprint, the trade-off in increased CPU / memory consumption is minimal.

Future work concerns further development of CEP-based distribution mechanisms for various monitoring use cases.

Acknowledgments. This work is partially supported by the European Union Regional Development Fund, POIG.02.03.00-00-007/08-00 as part of the PL-Grid Project.

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(4), 559–571 (2005)
2. Balis, B., Kowalewski, B., Bubak, M.: Real-time Grid monitoring based on complex event processing. Future Generation Computer Systems 27(8), 1103–1112 (2011), http://www.sciencedirect.com/science/article/pii/S0167739X11000562
3. Bolze, R., Caron, E., Desprez, F., Hoesch, G., Pontvieux, C.: A Monitoring and Visualization Tool and Its Application for a Network Enabled Server Platform. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3984, pp. 202–213. Springer, Heidelberg (2006)
4. Cooke, A., Gray, A.J.G., Ma, L., Nutt, W., Magowan, J., Oevers, M., Taylor, P., Byrom, R., Field, L., Hicks, S., Leake, J., Soni, M., Wilson, A., Cordenonsi, R., Cornwall, L., Djaoui, A., Fisher, S., Podhorszki, N., Coghlan, B.A., Kenny, S., O'Callaghan, D.: R-GMA: An Information Integration System for Grid Monitoring. In: Meersman, R., Schmidt, D.C. (eds.) CoopIS/DOA/ODBASE 2003. LNCS, vol. 2888, pp. 462–481. Springer, Heidelberg (2003)
5. Funika, W., Kryza, B., Slota, R., Kitowski, J., Skalkowski, K., Sendor, J., Krol, D.: Monitoring of SLA Parameters within VO for the SOA Paradigm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 115–124. Springer, Heidelberg (2010)
6. Gorla, A., Mariani, L., Pastore, F., Pezzè, M., Wuttke, J.: Achieving Cost-Effective Software Reliability Through Self-Healing. Computing and Informatics 29(1), 93–115 (2010)
7. 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)
8. 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)
9. Smith, M., Schwarzer, F., Harbach, M., Noll, T., Freisleben, B.: A Streaming Intrusion Detection System for Grid Computing Environments. In: Proc. HPCC 2009, pp. 44–51. IEEE Computer Society, Washington, DC (2009)
10. Truong, H.L., Fahringer, T.: SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. Scientific Programming 12(4), 225–237 (2004)
11. Wright, H., Crompton, R., Kharche, S., Wenisch, P.: Steering and visualization: Enabling technologies for computational science. Future Generation Computer Systems 26(3), 506–513 (2010)

Distributed Memory Virtualization with the Use of SDDSfL

Arkadiusz Chrobot, Maciej Lasota, Grzegorz Łukawski, and Krzysztof Sapiecha

Department of Computer Science, Kielce University of Technology, Poland
{a.chrobot,m.lasota,g.lukawski,k__sapiecha}@tu.kielce.pl

Abstract. Scalable Distributed Data Structures (SDDS) are a user–level software component that makes it possible to create a single coherent memory pool out of distributed RAMs of multicomputer nodes. In other words they are a tool for distributed memory virtualization. Applications that use SDDS benefit from a fast data access and a scalability offered by such data structures. On the other hand, adapting an application to work with SDDS may require significant changes in its source code. We have proposed an architecture of SDDS called SDDSfL that overcomes this difficulty by providing SDDS functionality for applications in a form of an operating system service. In this paper we investigate usefulness of SDDSfL for different types of applications.

Keywords: Scalable Distributed Data Structures, operating system, distributed memory virtualization.

1 Introduction

To be efficient, most of the software used in scientific or business data centers requires significant amounts of resources. The Random Access Memory (RAM) is an especially valuable resource. Keeping data working sets in the RAM allows applications to increase their efficiency by reducing the number of I/O operations [5]. A multicomputer is able to provide substantial quantities of the RAM for user applications. However, it is a system consisting of individual computers, so called nodes, connected through a fast local network [28] where each of the nodes has its own RAM. Therefore, a form of a virtualization [5,10,11] has to be used in order to create a unified, common RAM address space for the multicomputer.

Scalable Distributed Data Structures (SDDS) [14,17] are a software components that serve such a purpose. There are several variants of SDDS but all of them have some common properties [14]. SDDS usually store data in the RAM of multicomputer's nodes called servers. Other nodes, called clients, access the data. When a client accesses an item of data it computes the address of a server that possibly keeps this item. To this end it uses some parameters called a client's image. The address calculation procedure provides a fast access to data stored in the RAM of any SDDS server. Thus a common address space is created.

Every variant of SDDS must follow some fundamental design rules [17,19] that ensure the scalability and short data access time:

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 141–150, 2012.

1. No central directory is used in the algorithm of computing a data address.
2. The SDDS may change their size to adjust to current needs of clients.
3. The client's image may become outdated due to such changes but it is updated only when the client makes an addressing mistake.
4. No basic operation on data, such as inserting or deleting, requires an immediate and atomic image update of multiple clients.
5. Incorrectly addressed data is send to the right server and the client which made an addressing mistake receives an Image Adjusting Message (IAM) that updates its image.

The properties of SDDS are very valuable for multicomputer systems. Yet, SDDS are typical user–level software components, so adapting any application to work with them requires making significant changes in the source code of the application or at least developing a middleware that would connect the application to the SDDS.

To simplify that process we have proposed a new SDDS architecture for Linux (SDDSfL) [4]. In SDDSfL a client of SDDS is not a user–level software component but a part of an operating system. In this paper we describe the latest updates of the SDDSfL architecture and experimental results of an SDDSfL prototype implementation for a chosen set of centralized, user–level applications.

This paper is organized as follows. Section 2 contains a short summary of related work. In Section 3 a motivation for the research is given. The SDDSfL architecture is briefly described in Section 4. Results of an experimental evaluation of SDDSfL for different types of applications are presented in Section 5. The paper ends with conclusions.

2 Related Work

The problem of distributed memory virtualization is a major subject of many scientific works. This kind of virtualization may be applied for [5,25]:

1. extending a physical memory,
2. implementing a shared memory in distributed environments,
3. supporting applications that provide remote services.

The first goal is usually achieved by creating a distributed swap area for a standard virtual memory mechanism like demand paging [28] or by establishing a distributed RAM–disk. NSwap [20] and Distributed Anemone (DA) [8] are examples of a software that creates the distributed swap area. NSwap allows a client node to use the memory of one of the server nodes as a swap space. If the available memory size is not sufficient for the client's needs he may decide to migrate his pages to another server node. A client and a server communicate with each other using the UDP/IP protocol. DA uses its own low–level protocol for communication. Unlike NSwap it transparently increases the size of a swap space by adding the available memory of other multicomputer nodes when necessary. The Network Ram Disk (NRD) [7] makes it possible to use a free memory

of multicomputer nodes for creating a general purpose RAM–disk. Similarly to NSwap it uses the UDP/IP protocol for communication and stores client's data in only one server node at time.

The shared memory is one of inter–processes communication means that is available in multiprocessor systems with a common address space. In environments with no global address space such as a multicomputer or a NUMA multiprocessor the shared memory may be simulated by a Distributed Shared Memory DSM [21,22]. There are three ways of implementing the DSM: as a software, as a hardware or using a hybrid approach (mixed software and hardware). DASH [12] and DDM [29] are examples of a hardware implementation of the DSM, while IVY [13] and Clouds [1] represent the software approach. Flash [22] and ParaDiGM [2] belong to the hybrid solutions.

Applications that provide remote services tend to operate on working data sets that are larger than the available physical memory [5]. Applying a standard virtual memory scheme like the demand paging is the usual solution to such an issue. However, the virtual memory uses a hard disk as an additional memory space what increases the number of I/O operations and leads to longer data access time. In a multicomputer system this problem may be avoided by creating a single memory pool out of the available RAM of all nodes. Such an approach is offered by Oracle Coherence Data Grid [23] and products of RNA Network company [25].

There are some similarities between the DHT schemes like Chord [27], Symphony [18], CAN [24] and Pastry [26] and SDDS. However, the former ones address only the data location problem, while the SDDS are perfect data structures. [1]

3 Motivation

SDDS have many properties that are useful in the multicomputer system. They are scalable, reliable and provide short data access time. One of the limitations of the original SDDS design is its interface. It is inconvenient when porting a centralized application into a distributed environment. This task would require a significant changes in the source code of the application or at least an extra middleware that has to be developed. To avoid such an inconvenience the SDDS should use an interface which almost all applications are familiar with. That requirement justifies implementing SDDS as a part of the operating system. Another reason for implementing SDDS that way is their memory virtualization property. Hence, since the operating system is responsible for the memory management, the SDDS should be a part of it. In the next section we briefly describe a variant of SDDS, named SDDSfL [3,4], where a client is implemented as a block device driver, and so as a part of the operating system. Such an implementation

[1] The DHT schemes were designed to be used in P2P environments and they must cope with such issues as frequent joining and leaving of network nodes or even network partitioning. Such problems are less common in a multicomputer environment, and are not addressed in most of SDDS architectures. One of the exceptions is SDDS $\text{LH}^*_{\text{RS}}{}^{\text{P2P}}$ [15] which was especially designed for P2P environments.

of SDDS may be used as a substitute of a hard drive. It is expected that the more I/O requests an application performs the more it will profit from using SDDSfL. The goal of the research is to show when this claim is true, and to what extent.

4 Scalable Distributed Data Structure for Linux

We have designed and implemented a new architecture of SDDS for Linux operating system (SDDSfL) that allows the applications to use SDDS as a form of a system service. It is based on the SDDS LH* architecture which detailed description could be found in [17]. In this type of SDDS the client uses a distributed version of Linear Hashing algorithm [16] to find the addresses of SDDS servers. A record is a basic data unit in SDDS LH*. Records are grouped into lager data units called buckets which are kept in the RAM of SDDS servers. As a result of a record insertion some of the buckets may become overloaded. The SDDS deals with such an issue by splitting the buckets. The split operations are supervised by a SDDS component called a split coordinator (SC).

The main idea of the SDDSfL is that the client is implemented as a block device driver for Linux kernel. We made the decision basing on the basic properties of SDDS and block devices. Records in SDDS are identified by unique keys. Each data block which is the primary data unit of a block device has also a unique identifying number. SDDS store records without processing their content, the same way as the block devices. Other components of SDDSfL are implemented as a user–level applications that run on separate multicomputer nodes. The SDDSfL could be used by any application that uses a hard disk or a similar device, without any modifications. It may also serve as a swap device for the demand paging [28]. More details of SDDSfL architecture could be found in [4].

First experiences with SDDSfL have shown that running the whole client code in the main thread of the kernel makes the system prone to catastrophic failures. Therefore, we have made decision to divide the code of SDDSfL client into two parts. The fist part is directly responsible for handling the I/O operations issued by user–level applications. The second part is a separated kernel thread that takes care of network communication. It cooperates with the rest of SDDSfL client by exchanging data through a shared buffer. Such a separation of the I/O code and the networking code allows for better exceptions handling and reduces the risk of system crash. We have also made the SDDSfL client configurable from user–level with the use of sysfs [6].

5 Experimental Results

We have subjected the prototype implementation of SDDSfL to a series of tests in order to find out what kind of computations would benefit from using it. The main design goal of the SDDSfL is simplifying the migration of application's working data sets from a centralized environment to a distributed one. Therefore centralized, non-distributed programs were chosen for the experiments. In our

previous papers [3, 4] database applications and some preliminary numerical applications were tested. This work extends evaluation of the SDDSfL on a wider spectrum of I/O–bound and CPU–bound applications [28], summarizes the results and draws the conclusions from the research. We relate the results of SDDSfL to the performance of hard disks and distributed file systems.

In the first experiment the time of file sorting was measured. The QuickSort was used as an algorithm for file sorting. Because of that the program is an I/O–bound application. The tests were made for a set of files consisting of 2 KiB data records. The sizes of files range from 1 MiB to 1 GiB. The following hard disks were used for the tests: the SeaGate Barracuda 80 GB, 7200 rpm SATA disk and the SeaGate Cheetah 73GB, 15000 rpm SCSI Ultra–320 disk – both of them installed in a PC computer with 1.5 GiB of the RAM, the Maxtor 160 GB, 7200 rpm, Parallel ATA (PATA) UDMA/133 disk – installed in a PC computer with 2 GiB the RAM and the SeaGate Barracuda 160 GB, 7200 rpm SATA 2 disk installed in a multicomputer node with 2 GiB of the RAM. The SDDSfL ran on eight nodes of the same multicomputer equipped with two different local networks – InfiniBand (inf) and Gigabit Ethernet (gbe). The total size of SDDSfL file (the total capacity of all buckets) was 4 GiB. A QuickSort was used as an algorithm for the file sorting. Because of that the sorting program is an I/O–bound application. Figure 1 shows the results. As expected the SDDSfL allows the application to faster sort files consisting of small records than the majority of hard drives, even if it uses a relatively slow local network (Gigabit Ethernet).

We repeated the file sorting experiment for distributed file systems, namely Network File System (NFS) and LUSTRE, and for SDDSfL. This time we used two sets of files, one with files consisting of 2 KiB records and second with files

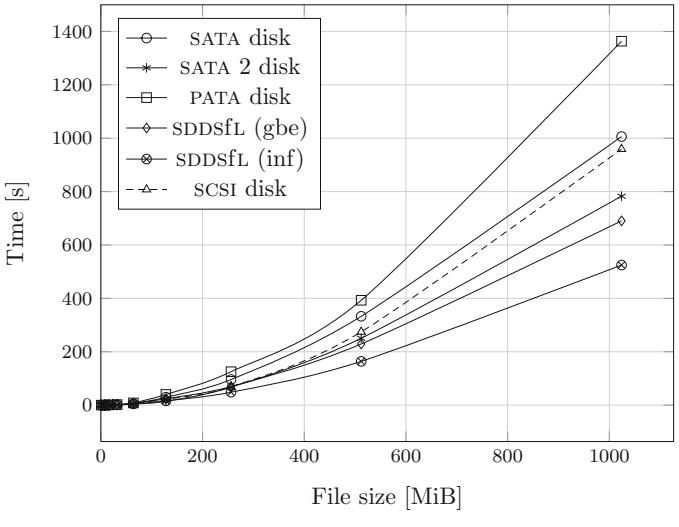


Fig. 1. Results of file sorting (record size = 2 KiB)

build of 4 KiB records. The tests were made with the use of Gigabit Ethernet network. Figure 2 shows the results. Number in parentheses represents the size of a record in bytes. In this test the SDDSfL is slightly better then NFS but they both outperform LUSTRE.

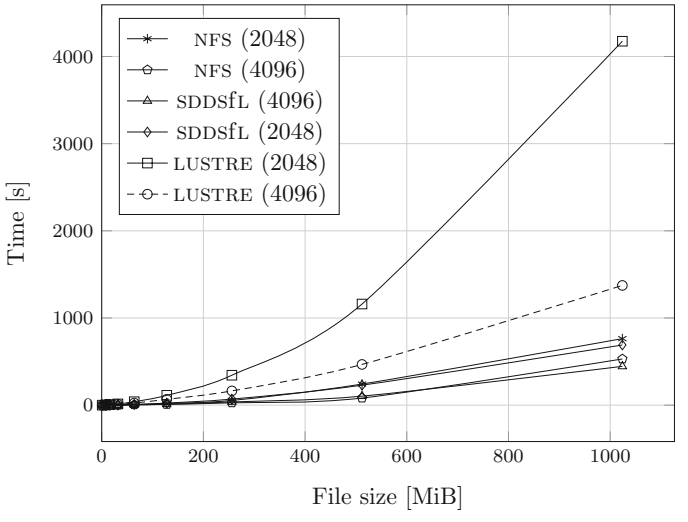


Fig. 2. Results of file sorting for DFS and SDDSfL

In the next experiment we measured the performance of a program that uses Boyer–Moore algorithm for pattern matching. Aside the previously described hard disks and DFS, a USB memory (pendrive) was used, namely the 8 GiB Kingston Data Traveler. If a single, short pattern is searched in relatively large files, the efficiency of such a process depends mostly on the I/O performance (Figure 3). However, such a task requires only a sequential access to data, so the random access oriented SDDSfL may be outperformed by hard drives in case of big files. If several patterns are searched in a single run the task becomes CPU–bound (Figure 4) and the performance of I/O requests becomes meaningless for its efficiency.

In the last experiment the SDDSfL was used as a swapping device in a multicomputer created from ordinary PC computers with 2 GiB of the RAM each one. Its performance was compared to the results of the SeaGate Barracuda 80 GB, 7200 rpm, SATA 2 hard disk. A special program was used for this test that allocates a memory area of a size equal to the size of the physical memory and gradually writes zeros to this area. In each iteration the process of writing starts from the beginning of the area and increases by 1 MiB. At some point the operating system starts to swap pages to the external memory. The results in Figure 5 show that when the activity of swapping becomes intensive the SDDSfL

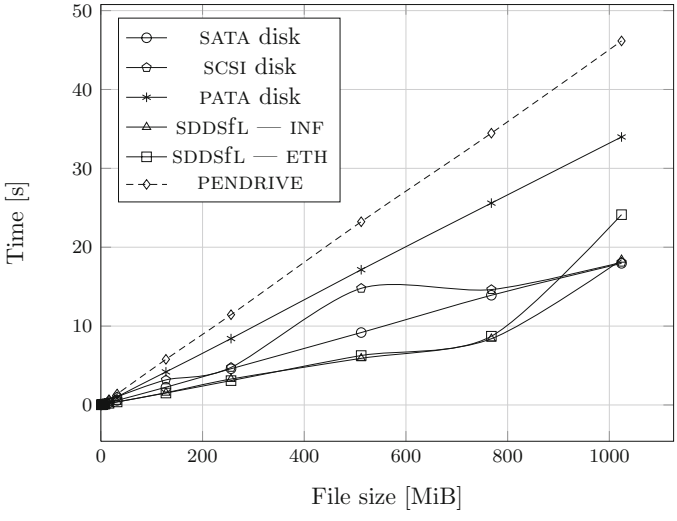


Fig. 3. Pattern matching (a 1024 characters long pattern)

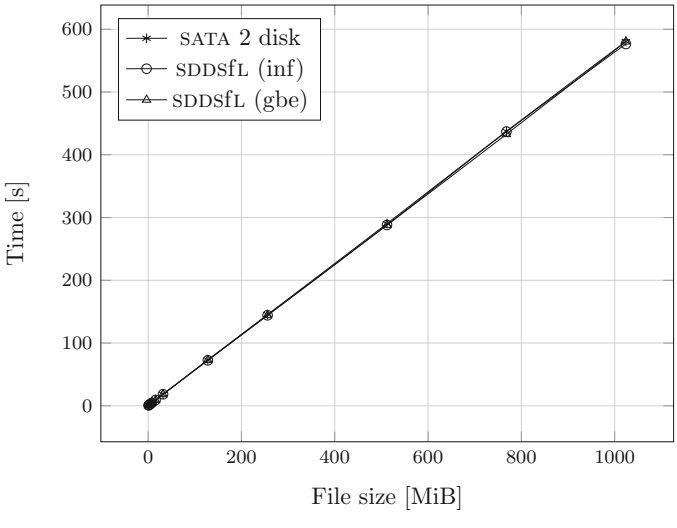


Fig. 4. Pattern matching (several patterns)

works better as a swap device than the hard drive, because of its random access nature. The obtained results are comparable to the results of Distributed Anemone described in [8].

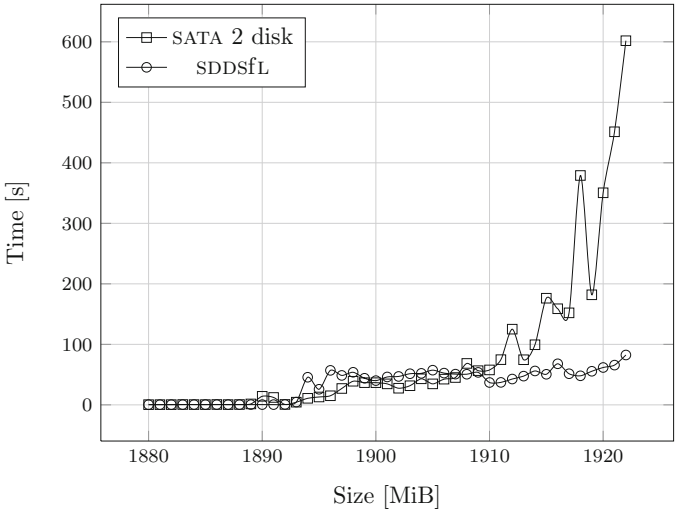


Fig. 5. Page swapping results

6 Conclusions

The Scalable Distributed Data Structure for Linux (SDDSfL) is a software tool for distributed memory virtualization. Similarly to Oracle Coherence Data Grid it changes the I/O–operations into RAM references, but leaves a hard disk interface for applications.

Results of the experiments described in Section 5 together with results previously presented in [3, 4] show that I/O–bound applications, like transactional databases, generally perform their tasks more efficiently when they use SDDSfL rather than a hard disk. However, modern hard disks are equipped with their own hardware cache memory, which reduces the time needed for performing sequential I/O–operations [9]. The cache combined with a fast bus, like SATA 2, allows contemporary hard disks to get closer to the results of the SDDSfL, provided that the applications require a sequential access to data and that the SDDSfL uses Gigabit Ethernet or a similar local network. In case of applications that require a random data access, like the page swapping, the hard disk cache becomes inefficient, hence better results of the SDDSfL.

When compared to distributed file systems the SDDSfL outperforms LUSTRE file systems and gives similar results to NFS for I/O–bound applications. All the three solutions were tested with the use of a Gigabit Ethernet network. It is worth to mention that the Network File System (NFS) is not a truly distributed file system. It allows many clients to access a single hard disk, so its performance is similar to the performance of a local hard disk [2]. The efficiency of LUSTRE for

[2] NFS v. 4.1 permits a client to access more than one disk in parallel. However, this version was not used in our tests.

I/O–bound applications is very low. The possible cause for this is that the work of hard disks installed on separated nodes of multicomputer is not synchronized.

For the CPU–bound applications the differences in performance of all tested solutions are negligible. The overall conclusion is following: the more random I/O–operations an application does the more it benefits from using the SDDSfL.

In the experiments the SDDSfL cooperated with the InfiniBand network through a TCP/IP protocol stack, which hampered its overall efficiency. A direct communication with the use of the native InfiniBand protocols should further improve the performance.

Although the SDDSfL prototype implementation was developed for Linux operating system its architecture is to some degree universal. It may be adapted for any operating system that supports the block device paradigm. All applications that uses hard disks or similar devices may use the SDDSfL almost at once, without expensive modifications of their source code. The only changes, that might be necessary, apply only to environmental variables or configuration files of applications and could be performed by a system operator without any assistance of a software engineer.

References

1. Chen, R.C., Dasgupta, P.: Implementing consistency control mechanisms in the Clouds distributed operating system. In: ICDCS, pp. 10–17 (1991)
2. Cheriton, D., Goosen, H.A., Boyle, P.D.: ParaDiGM: A Highly Scalable Shared-Memory Multi-Computer Architecture. IEEE Computer 24, 33–46 (1991)
3. Chrobot, A., Lasota, M., Lukawski, G., Sapiecha, K.: SDDSfL vs. local disk — a comparative study for Linux. Annales UMCS Informatica 10, 29–39 (2010)
4. Chrobot, A., Lukawski, G., Sapiecha, K.: Scalable Distributed Data Structures for Linux-based Multicomputer. In: International Symposium on Parallel and Distributed Computing, pp. 424–428. IEEE Computer Society (2008)
5. Cook, C.: Memory Virtualization, the Third Wave of Virtualization, http://vmblog.com/archive/2008/12/14/memory-virtualization-the-third-wave-of-virtualization.aspx
6. Corbet, J., Rubini, A., Kroah-Hartman, G.: Linux Device Drivers, 3rd edn. O'Reilly Media, Inc. (2005)
7. Flouris, M.D., Markatos, E.P.: The Network RamDisk: Using Remote Memory on Heterogeneous NOWs. Cluster Computing 2, 281–293 (1999)
8. Hines, M.R., Wang, J., Gopalan, K.: Distributed Anemone: Transparent Low-Latency Access to Remote Memory. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2006. LNCS, vol. 4297, pp. 509–521. Springer, Heidelberg (2006)
9. Hsu, W., Smith, A.J.: The performance impact of I/O optimizations and disk improvements. IBM J. Res. Dev. 48(2), 255–289 (2004)
10. Kusnetzky, D.: RNA Networks and memory virtualization, http://blogs.zdnet.com/virtualization/?p=655
11. Kusnetzky, D.: Sorting out the different layers of virtualization, http://blogs.zdnet.com/virtualization/?p=170
12. Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W., Gupta, A., Hennessy, J., Horowitz, M., Lam, M.S.: The Stanford DASH multiprocessor. IEEE Computer 25, 63–79 (1992)

13. Li, K., Hudak, P.: Memory coherence in shared virtual memory systems. ACM Trans. Comput. Syst. 7(4), 321–359 (1989)
14. Litwin, W., Neimat, M.A., Schneider, D.: RP*: A Family of Order Preserving Scalable Distributed Data Structures. In: Proceedings of the Twentieth International Conference on Very Large Databases, Santiago, Chile, pp. 342–353 (1994)
15. Litwin, W.: LH*RS P2P: A Scalable Distributed Data Structure for P2P Environment, http://video.google.com/videoplay?docid=-7096662377647111009#
16. Litwin, W.: Linear hashing: a new tool for file and table addressing. In: VLDB 1980: Proceedings of the Sixth International Conference on Very Large Data Bases, pp. 212–223. VLDB Endowment (1980)
17. Litwin, W., Neimat, M.A., Schneider, D.A.: LH* — a scalable, distributed data structure. ACM Transactions on Database Systems 21(4), 480–525 (1996)
18. Manku, G.S., Bawa, M., Raghavan, P., Inc, V.: Symphony: Distributed Hashing in a Small World. In: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, pp. 127–140 (2003)
19. Ndiaye, Y., Diene, A., Litwin, W., Risch, T.: AMOS-SDDS: A Scalable Distributed Data Manager for Windows Multicomputers. In: 14th Intl. Conference on Parallel and Distributed Computing Systems – PDCS 2001 (2001)
20. Newhall, T., Finney, S., Ganchev, K., Spiegel, M.: Nswap: A Network Swapping Module for Linux Clusters. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 1160–1169. Springer, Heidelberg (2003)
21. Nitzberg, B., Lo, V.: Distributed Shared Memory: A Survey of Issues and Algorithms. Computer 24(8), 52–60 (1991)
22. Protic, J., Tomasevic, M., Milutinovic, V.: A survey of distributed shared memory systems. In: Hawaii International Conference on System Sciences, pp. 74–84 (1995)
23. Purdy, C.: Getting Coherence: Introduction to Oracle Coherence Data Grid, http://www.youtube.com/watch?v=4Sq45B8wAXc
24. Ratnasamy, S., Francis, P., Shenker, S., Karp, R., Handley, M.: A Scalable Content-Addressable Network. In: Proceedings of ACM SIGCOMM, pp. 161–172 (2001)
25. RNA Networks: Memory Virtualization Primer, http://www.rnanetworks.com/cache-architecture
26. Rowstron, A., 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, pp. 329–350. Springer, Heidelberg (2001)
27. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: Proceedings of the 2001 ACM SIGCOMM Conference, pp. 149–160 (2001)
28. Tanenbaum, A.S.: Modern Operating Systems. Pearson Prentice Hall, Upper Saddle River (2008)
29. Warren, D.H.D., Haridi, S.: Data Diffusion Machine - A Scalable Shared Virtual Memory Multiprocessor. In: FGCS, pp. 943–952 (1988)

Dynamic Compatibility Matching of Services for Distributed Workflow Execution

Paweł Czarnul and Michał Wójcik

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Poland
{pczarnul,michal.wojcik}@eti.pg.gda.pl

Abstract. The paper presents a concept and an implementation of dynamic learning of compatibilities of services used in a workflow application. While services may have the same functionality, they may accept input and produce output in different formats. The proposed solution learns matching of outputs and inputs at runtime and uses this knowledge in subsequent runs of workflow applications. The presented solution was implemented in an existing workflow execution system – BeesyBees.

Keywords: matching of services, workflow application, workflow execution, dynamic learning.

1 Introduction

Traditionally, a complex task can be defined as a workflow application which is modelled as an acyclic directed graph $G = (V, E)$ in which vertices $V = \{t_1, t_2, \ldots, t_{|V|}\}$ correspond to subtasks of the original task while edges $E = \{e_1, e_2, \ldots, e_{|E|}\}$ denote time dependencies between the subtasks [1]. For each of the subtasks, a different service from a different provider can be assigned. Integration of distributed services into workflow applications is associated with the following steps: definition of a complex task, searching for services capable of subtasks, QoS selection of services for subtasks, execution and reselection of services if conditions have changed or previously selected services have failed.

Even though services may be able to perform subsequent parts of the given complex task, it is necessary to determine that the output of a given service can be understood properly by a service that follows. Consequently, learning and knowledge about matching of inputs and outputs of services considered in this work allow to determine particular services that would accept outputs of previous ones. It is a technique that allows a more precise search in step 1 above.

2 Problem Statement and Related Work

Firstly, for definition of a complex workflow functional specification of particular subtasks needs to be defined.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 151–160, 2012.

Secondly, services capable of executing particular subtasks need to be found. For instance, a service description in OWL-S [2] can specify its functions in the `service Category` element. Descriptions of services can be matched with functional descriptions of subtasks. This can be done through ontologies which link concepts from both descriptions using e.g. the inheritance relation [3], [4]. Systems such as Meteor-S [5] allow for specification of both functional and non-functional descriptions and finding services.

Thirdly, for each subtask, one service needs to be chosen so that a global QoS criterion is optimized while meeting other QoS constraints. For instance, the goal is to minimize the workflow execution time while keeping the cost of selected services below the given budget [6]. Algorithms such as integer linear programming, genetic algorithms, GAIN etc. can be used for this purpose [1].

Furthermore, should a failure occur in the service or new services appear, dynamic reselection of services is applied [7].

There exist many functional service descriptions available in UDDI registries [8] or Web-based registries of services such as `www.service-repository.com`, `www.xmethods.net/ve2/index.po`, `www.embraceregistry.net` or `webservices.washington.edu`. Similarly, a previous approach of adopting Linux applications to services [9] uses descriptions of the functions performed by the applications from the man pages or descriptions available in rpm or deb packages. However, there arises a problem with matching output data formats of a service with input data formats of following services. The goal of this work is to construct an automatic mechanism for learning compatibilities of outputs and inputs of services and incorporation into future service selection decisions. Namely, apart from selection of services capable of executing a particular function, such services are selected so that their allowed inputs are compatible with outputs of already executed and preceding services.

3 Proposed Solution

3.1 System Architecture

The idea presented in this paper was implemented in an existing workflow execution system, BeesyBees [10,11] which is a module of BeesyCluster [12].

BeesyCluster can be regarded as a middleware for integrating and accessing resources like files and services located on various distributed servers and clusters. It allows its users to construct, invoke and execute complex workflows, prepared with the provided editor and using services available to the workflow creator. Services can be chosen both from those published by the user modelling the workflow as well as made available by other system users [7].

The BeesyBees system (its architecture is presented in Figure 1) was prepared in order to add a decentralized and distributed workflow execution module based on autonomous agents to the BeesyCluster. It is based on the JADE (Java Agent DEvelopment Framework) [13].

The system consists of a number of nodes (called containers in JADE) where agents can reside and migrate between them. All agents inside the system are

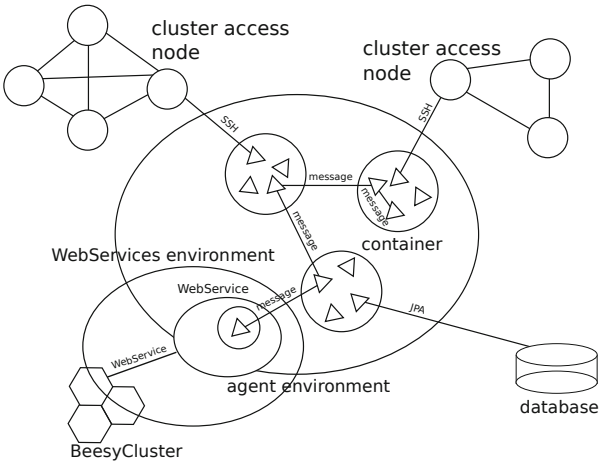


Fig. 1. Architecture of BeesyBees

able to communicate with each other using ACLMessages. Agents are capable of executing workflow services, which according to BeesyCluster documentation are programs called through the SSH protocol. All the data needed between different executions are stored persistently in a database.

There are four types of agents essential for this work:

- GateWayAgent — an agent running in a Web Service deployed in a Java EE server to mediate in communication between the BeesyBees and external systems e.g. BeesyCluster. Allows for adding new and cancelling existing workflows as well as notifications about progress of workflow execution.
- StatusAgent — agents responsible for monitoring an execution process of all workflows delegated to BeesyBees. They decide which workflows should be accepted and launched. They gather information about execution progress and persist it in order to be able restart execution from the moment when it was interrupted e.g. because of a system failure.
- RepositoryAgent — an agent responsible for storing information concerning services used in workflows execution. Each agent can store and fetch some information describing QoS of services.
- TaskAgent — A group of TaskAgents execute the workflow cooperatively using negotiations [14]. They are able to distribute tasks between each other in a decentralized way. Moreover, when one of them fails and disappears from the system, another is able to pick an orphaned task.

3.2 Proposed Algorithm

It is assumed that the following descriptions of services are available to the author of the workflow:

1. a service with a functional description without semantic descriptions of input and output,
2. a service with a full description contained in e.g. OWL-S,
3. a service with only input or output described in e.g. OWL-S.

In all cases, a description of input or output does not imply that such a description is complete. This results from the fact that for services with only a functional description, input/output specification is generated during system runtime.

Secondly, we consider the following types of services with respect to handling input and output formats:

- fixed formats e.g. able to accept only one format and returning output in only one specified format,
- able to accept and/or return output in many different formats. As an example, ImageMagick's `convert` command is able to process and produce various image formats (i.e. JPEG, PNG,GIF) depending on program input.

An important factor is the way to determine if the input of a service is compliant with the output(s) of previous service(s); this can be accomplished by e.g. checking that the size of the output is not smaller than a specified threshold. In the implementation used by the authors, the exit status of a program representing the service was examined. It is assumed that it is different from 0 for programs which have failed.

This paper assumes service comparison based on the specification of input and output handled by the services. It is appropriate to compare not only data types but also additional formatting requirements (e.g. a service may accept on input integers between 0 and 256); the specification of such requirements is possible and supported by OWL-S [2] semantic description.

OWL-S descriptions consist of three top level values: (i) a service profile describing what a service does for a service-seeking agent, (ii) service grounding that specifies technically how the service can be accessed, and (iii) a service model that explains in detail how to interact with the service and what output is generated based on the input.

For the purposes of this work only the service profile is used. This is because it contains details what types of input (`hasInput` parameter) and output (`hasOutput` parameter) are handled by the described service. Each of the inputs and outputs are described as RDF [15] entities denoting the type and format of those parameters. Additionally, the profile provides a human readable service description (serviceClassification, serviceProduct, serviceName, and textDescription parameters) and a service category allowing for categorization and recognition.

The following algorithm for workflow execution is proposed by the authors. The following are assumed:

1. a matrix that denotes matching between service s_{ij} and s_{kl}; $MS(s_{ij}, s_{kl})$ denotes the matching score between the two services at the given moment. s_{ij} means the j-th service capable of executing task t_i. It is assumed that

the higher MS the better the matching. Range $[0, 1]$ is assumed. Namely, this is defined as $MC(s_{ij}, s_{kl})/C(s_{ij}, s_{kl})$ in which MC denotes the number of cases in which output of service s_{ij} matched input of service s_{kl} until now where C denotes the number of successive runs of these services;

2. a description of each service s_{ij} that contains the following sets of fields:
   - set $I(s_{ij})$ of fields that specify inputs accepted by the service,
   - set $O(s_{ij})$ of fields that specify outputs that could be produced by the service.

   As mentioned above, this data may be given for some services, may be given partially or not available at all when the system starts its operation.

The algorithm proceeds with the steps shown in Figure 2. For each task to be executed, a list of services capable of executing the task is created. Then, a selection algorithm is applied that selects the best service to be executed considering both matching outputs and inputs of services. This is followed by execution of the selected service and a learning phase that learns about matching of outputs and inputs of services so that this knowledge can be reused in the selection phase in the future.

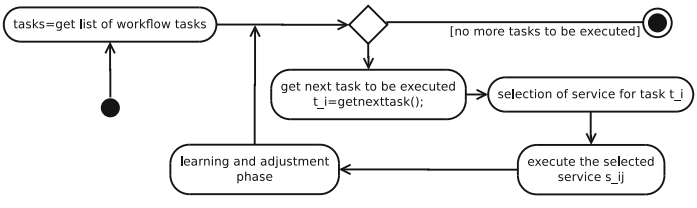


Fig. 2. Activity Diagram for the Matching Algorithm

The service selection phase described in Listing 1.1 takes all services assigned to current task t_k (set S_k) and the services chosen for execution of all preceding tasks (SP_k). In order to provide learning about unknown services and updating knowledge (some services can change their properties at runtime) a probability threshold $THRE\ SHOLD$ is used to determine whether to select a service randomly (to learn about it) or rely on a selection based on the previous knowledge about outputs and inputs of services. In the latter case, for each service in S_k a matching score is calculated. The result consists of matching scores between preceding services $s_{ij} \in SP_k$ and the considered service s_{kl}. Additionally, the minimum value of matching scores multiplied by 1000 is added. This is to express that the minimum matching score between the current and previous service is crucial in evaluation of the given service.

Individual matching scores are calculated according to a function presented in Listing 1.2. In situations when both services have defined descriptions in OWL-S (it is enough for s_{kl} to have $I(s_{kl})$ defined and s_{ij} to have $O(s_{ij})$ defined) their compatibility is verified by checking if all output descriptions of s_{ij} form

a subset of all input descriptions of s_{kl}. In cases where a service lacks required description, the learned matching scores are returned. Additionally, in order to avoid mistakes resulting from a small number of calls ($C(s_{ij}, s_{kl})$), the matching score is always maximum for services for which the numbers of runs are below a defined threshold ($C\_THRESHOLD$).

For services with equal matching scores, additional conditions should be taken into account. For instance, a weighted score can be used that incorporates both compatibility matching and service QoS [6]. In cases when all conditions are equal, a service can be chosen randomly from those with highest matching scores. The experiments in this work focus on the compatibility matching.

Listing 1.1. Get Best Service

```
function get_best(SP_k, S_k) : Service
begin
  p := rand(0,1)
  if p < THRESHOLD
    return s_kl : s_kl ∈ S_k ∧ l = rand(1, |S_k|)
  end
  for s_kl in S_k
    mscore_{s_kl} := 1000 · min_{s_ij ∈ SP_k}(score(s_ij, s_kl)) + Σ_{s_ij ∈ SP_k} score(s_ij, s_kl)
  end
  return s_kl : s_kl ∈ S_k ∧ mscore_{s_kl} = max_l mscore_{s_kl}
end
```

Listing 1.2. Compatibility Scores between Pairs of Services

```
function score(s_ij, s_kl) : double
begin
  if I(s_kl) ≠ ∅ and O(s_ij) ≠ ∅ and O(s_ij) ⊂ I(s_kl)
    return 1
  else if C(s_ij, s_kl) < C_THRESHOLD
    return 1
  else
    return MS(s_ij, s_kl)
  end end
```

Listing 1.3. Learning about Compatibility between Services

```
function learning(SP_k, s_kl, success)
begin
  for s_ij in SP_k
    C(s_ij, s_kl)++
    if success(s_ij, s_kl) = true
      MC(s_ij, s_kl)++
      if MS(s_ij, s_kl) == 1 and C(s_ij, s_kl) > M_THRESHOLD
        I(s_kl) := I(s_kl) ∪ O(s_ij)
      end
    else
      if I(s_ij) ≠ ∅ and O(s_ij) ≠ ∅ and O(s_ij) ⊂ I(s_ij)
        intersection := O(s_ij) ∩ I(s_kl)
        O(s_ij) := O(s_ij) \ intersection
        I(s_kl) := I(s_kl) \ intersection
      end end end end
```

Listing 1.3 presents the process of learning about service compatibility after a particular task has been executed. Parameters taken by the function are all preceding services (SP_k), the currently executed service (s_{kl}) and flags indicating

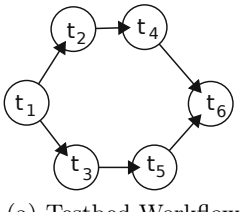
if the service was executed successfully for the output data passed from service s_{ij} $(success(s_{ij}, s_{kl}))$.

For all preceding services ($s_{ij} \in SP_k$) the number of successive calls is incremented. The number of matching calls is increased only in the cases of successful task execution for the data passed from the particular preceding service. Descriptions made in OWL-S are being updated after successful execution in cases when the matching score is maximum and the number of successive calls is above a predefined threshold ($M\_THRESHOLD$). In case of failures and if selection was made on the basis of OWL-S descriptions, descriptions are updated irrespective of the number of calls.

4 Experiments

Figure 3a presents a testbed workflow that consists of six tasks and two parallel paths. For each task three different services were prepared. Those services differ in input/output specification, that is they accept and produce different formats of input and output parameters respectively. Figure 3b shows detailed specification of accepted inputs and produced outputs by all services, where a set of all possible formats is $\{a, b, c, d, e, f, g, h, i\}$. Figure 4a shows a matrix with real matching scores that the algorithm ideally should learn. This shows only pairs of services which are 100% compatible that is any output is accepted by the following service. This shows pairs which are then tested in testbed workflow. For test purposes, all services have output described semantically but no information about input. All tests were performed using a distributed environment with multiple agents, agent system nodes and service servers. During execution each task agent was programmed to pick a service up to six times (in case if the selected service appeared incompatible with the preceding ones selected previously). Workflow execution was cancelled beyond this value. The probability of selection a service randomly was set to $THRESHOLD = 0.3$.

In order to show the way the algorithm learns, the same workflow was run 400 times. Each time different formats of initial inputs were used. At first, all matching scores are equal which means that the algorithm does not have any



(a) Testbed Workflow

I (s11) = {a, b, c, d, e, f , g, h, i } O(s11) = {a, g, h, i }	I (s21) = {a, b, g, h, i } O(s21) = {a, g, h, i }	I (s31) = {a, g, h, i } O(s31) = {d, g, h, i }
I (s12) = {a, b, c, d, e, f , g, h, i } O(s12) = {b, g, h, i }	I (s22) = {b, g, h, i } O(s22) = {b, g, h, i }	I (s32) = {b, c, g, h, i } O(s32) = {e, g, h, i }
I (s13) = {a, b, c, d, e, f , g, h, i } O(s13) = {c, g, h, i }	I (s23) = {c, g, h, i } O(s23) = {c, g, h, i }	I (s33) = {c, g, h, i } O(s33) = {f , g, h, i }
I (s41) = {a, c, g, h, i } O(s41) = {a, g, h, i }	I (s51) = {d, e, g, h, i } O(s51) = {d, g, h, i }	I (s61) = {a, d, f , g, h, i } O(s61) = {a, g, h, i }
I (s42) = {b, g, h, i } O(s42) = {b, g, h, i }	I (s52) = {e, g, h, i } O(s52) = {e, g, h, i }	I (s62) = {a, b, c, e, g, h, i } O(s62) = {b, g, h, i }
I (s43) = {b, c, g, h, i } O(s43) = {c, g, h, i }	I (s53) = {e, f , g, h, i } O(s53) = {f , g, h, i }	I (s63) = {c, d, f , g, h, i } O(s63) = {c, g, h, i }

(b) Specification of Original Inputs/Outputs for Services

Fig. 3. Testbed Environment

oi	s_{21}	s_{22}	s_{23}	s_{31}	s_{32}	s_{33}	s_{41}	s_{42}	s_{43}	s_{51}	s_{52}	s_{53}	s_{61}	s_{62}	s_{63}
s_{11}	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
s_{12}	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
s_{13}	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
s_{21}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
s_{22}	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
s_{23}	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
s_{31}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
s_{32}	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
s_{33}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
s_{41}	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
s_{42}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_{43}	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
s_{51}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
s_{52}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_{53}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

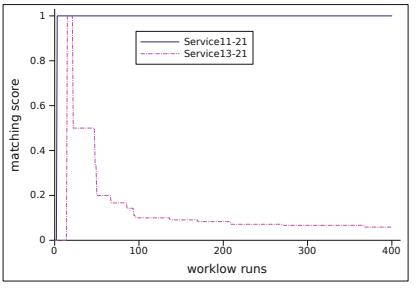
(a) Real (Designed) and Expected Matching Scores

oi	s_{21}	s_{22}	s_{23}	s_{31}	s_{32}	s_{33}	s_{41}	s_{42}	s_{43}	s_{51}	s_{52}	s_{53}	s_{61}	s_{62}	s_{63}
s_{11}	1	0	0	1	0	0	1	0	0	0	0	0	1	1	0
s_{12}	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
s_{13}	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1
s_{21}	1	0	0	1	0	0	1	0	0	0	0	0	1	1	0
s_{22}	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0
s_{23}	0	0	1	0	0	1	1	0	1	0	0	0	0	0	1
s_{31}	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
s_{32}	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
s_{33}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
s_{41}	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0
s_{42}	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0
s_{43}	0	0	1	0	0	1	0	0	0	0	0	0	0	1	1
s_{51}	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1
s_{52}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
s_{53}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
s_{61}	1	0	0	1	0	0	1	0	0	0	0	0	1	1	0
s_{62}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_{63}	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1

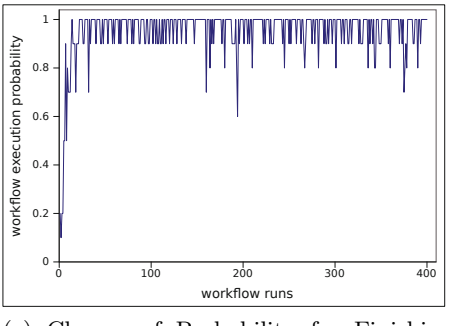
(b) Compatible Services according to Semantic Descriptions after 400 Runs

oi	s_{21}	s_{22}	s_{23}	s_{31}	s_{32}	s_{33}	s_{41}	s_{42}	s_{43}	s_{51}	s_{52}	s_{53}	s_{61}	s_{62}	s_{63}
s_{11}	1	0.11	0.06	1	0.08	0.07	0	0	0	0	0	0	0	0	0
s_{12}	1	1	0.09	0.02	1	0.05	0	0	0	0	0	0	0	0	0
s_{13}	0.07	0.01	1	0.06	1	1	0	0	0	0	0	0	0	0	0
s_{21}	0	0	0	0	0	0	1	0.11	0	0	0	0	0	0	0
s_{22}	0	0	0	0	0	0	0.17	1	1	0	0	0	0	0	0
s_{23}	0	0	0	0	0	0	1	0.09	1	0	0	0	0	0	0
s_{31}	0	0	0	0	0	0	0	0	0	1	0.17	0.06	0	0	0
s_{32}	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
s_{33}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
s_{41}	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0.11
s_{42}	0	0	0	0	0	0	0	0	0	0	0	0	0.17	1	0.79
s_{43}	0	0	0	0	0	0	0	0	0	0	0	0	0.19	1	1
s_{51}	0	0	0	0	0	0	0	0	0	0	0	0	1	0.12	1
s_{52}	0	0	0	0	0	0	0	0	0	0	0	0	0.21	1	0.29
s_{53}	0	0	0	0	0	0	0	0	0	0	0	0	1	0.04	1

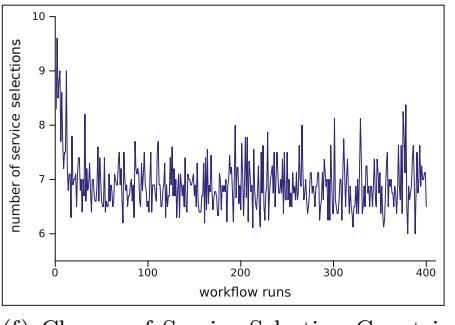
(c) Matching Scores after 400 Runs



(d) Change of Matching Score in Workflow Runs



(e) Change of Probability for Finishing Worklow in Runs



(f) Change of Service Selection Count in Workflow Runs

Fig. 4. Tests Results

knowledge. The matrix shown in Figure 4c shows updated matching scores at the end of all runs. Figure 4d shows how during subsequent runs the matching scores between pairs $\{s_{11}, s_{21}\}$ and $\{s_{13}, s_{21}\}$ change converging to 1 and 0 respectively. Figure 4e shows the probability of finishing a workflow successfully which grows while gaining knowledge about services. The fact that it does not reach 1 stems

from the fact that occasionally a service is chosen randomly. Figure 4f shows the number of service selections which decreases in subsequent runs. Fluctuations in results are caused by the $THRESHOLD$ value. The ideal value for the testbed workflow is equal to the number of nodes i.e. 6.

The matrix in Figure 4b presents which services are regarded as compatible according to the updated semantic descriptions. Values of matching scores are leading to designed values. Compatibility between pairs of services unchecked during the test was detected. This can be useful for running new workflows. For example, services s_{61} and s_{21} were not connected in the testbed workflow. Service s_{61} has output described as $O(s_{61}) = \{a, g, h, i\}$. Thanks to testing connections between services s_{11}, s_{12} and s_{21} it has been learned that the input for s_{21} is $I(s_{21}) = \{a, b, g, h, i\}$ which implies compatibility between s_{61} and s_{21}.

5 Conclusions and Future Work

This work presents a decentralized and distributed system for execution of workflows using autonomous agents. The presented method for selection of an appropriate service for a particular task is based on compatibility between pairs of successive services. The presented algorithm uses two ways for comparison of services: (i) on the basis of semantic descriptions of input and outputs handled by services, and (ii) using matching scores between pairs of services describing statistically successful successive calls.

Additionally, the algorithm introduces a learning phase allowing for updates of an actual value of a matching score and updating semantic descriptions after a predefined number of successful successive calls.

Currently, two services are compatible according to semantic description iff descriptions of all outputs of a service form a subset of all descriptions of inputs of another service. In the future, the system should be able to reason how data flowing through services change in terms of formats in order to choose services for which input and output sets descriptions have some intersection without a need for a complete inclusion as above. Namely, knowing inputs, the algorithm is able to determine possible outputs and consequently inputs and outputs of successive services. Moreover, it would improve the process of updating semantic descriptions both in cases of successful and unsuccessful task execution. Furthermore, the approach will be tested on larger real world scenarios designed and deployed at the Faculty of ETI, Gdańsk University of Technology along with tests when scaling the system up.

Acknowledgements. Work supported in part by the Polish Ministry of Science and Higher Education under research project number N N519 172337 titled "Integration-Oriented Method of Application Development with High Dependability Requirements".

References

1. Yu, J., Buyya, R., Ramamohanarao, K.: Metaheuristics for Scheduling in Distributed Computing Environments. In: Workflow Scheduling Algorithms for Grid Computing. Springer (2008)
2. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services (2004)
3. Srinivasan, N., Paolucci, M., Sycara, K.: Adding Owl-s to Uddi, Implementation and Throughput. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
4. Czarnul, P.: Wykorzystanie ontologii do wyszukiwania usług w systemie BeesyCluster. In: KASKBOOK 2009, Politechnika Gdańska (2009)
5. Aggarwal, R., Verma, K., Miller, J., Milnor, W.: Constraint driven web service composition in METEOR-S. In: Proceedings of IEEE International Conference on Services Computing (SCC 2004), pp. 23–30 (2004)
6. Yu, J., Buyya, R., Tham, C.K.: Cost-based scheduling of workflow applications on utility grids. In: Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science 2005). IEEE CS Press, Melbourne (2005)
7. Czarnul, P.: Modeling, run-time optimization and execution of distributed workflow applications in the JEE-based BeesyCluster environment. The Journal of Supercomputing, 1–26 (2010)
8. Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., et al.: Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI. SAMS Publishing (2002)
9. Czarnul, P., Kurylowicz, J.: Automatic conversion of legacy applications into services in beesycluster. In: 2nd International Conference on Information Technology (ICIT), pp. 21–24 (2010)
10. Czarnul, P., Matuszek, M., Wójcik, M., Zalewski, K.: BeesyBees - Agent-Based, Adaptive & Learning Workflow Execution Module for BeesyCluster. In: Faculty of ETI Annals, Information Technologies, vol. 18. Gdańsk University of Technology (2010)
11. Czarnul, P., Matuszek, M., Wójcik, M., Zalewski, K.: BeesyBees - Efficient and Reliable Execution of Service-based Workflow Applications for BeesyCluster using Distributed Agents (BEST PAPER). In: Proceedings of the 2010 International Multiconference on Computer Science and Information Technology (IMCSIT), vol. 5, pp. 173–180 (2010)
12. Czarnul, P., Bajor, M., Frączak, M., Banaszczyk, A., Fiszer, M., Ramczykowska, K.: Remote Task Submission and Publishing in BeesyCluster: Security and Efficiency of Web Service Interface. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 220–227. Springer, Heidelberg (2006)
13. Telecom Italia Lab: JADE (Java Agent DEvelopment Framework) online documentation (2011), http://jade.tilab.com/doc/index.html
14. Wójcik, M.: Effective decisions making in distributed agent systems. Master's thesis, Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics (2010)
15. Manola, F., Miller, E.: Rdf primer, w3c recommendation (2004)

Cyberinfrastructure Support for Engineering Virtual Organization for CyberDesign

Tomasz Haupt, Nitin Sukhija, and Mark F. Horstemeyer

Mississippi State University, Center for Advanced Vehicular Systems,
Starkville, MS 39759, USA
{haupt,nitin,mfhorst}@cavs.msstate.edu

Abstract. Integrated Computational Material Engineering (ICME) is an emerging discipline transforming materials science. Computational engineering accelerates materials development, integrates design and manufacturing, and unifies these with the engineering design optimization process, as well as efficiently employs greater accuracy in simulation-based design. Efforts to realize this enormous and complex goal have catalyzed the development of the Engineering Virtual Organization for Cyber Design (EVOCD), which provides a cyberinfrastructure to accumulate and protect the intellectual property pertaining to selected aspects of materials science and engineering that is generated by the participants of the organization, to enforce the quality of that information, and to manage its complexity. The intellectual property includes experimental data, material models and constants, computational tools and software artifacts, and the knowledge pertaining to multiscale physics-based models for selected properties and processes. EVOCD has been developed using open source components augmented with custom modules such as a secure data repository integrated with online model calibration tools. EVOCD is available at http://icme.hpc.msstate.edu

Keywords: Virtual Organizations, Knowledge Management, Cyberinfrastructure, Service-Oriented Software Engineering, Data Repository.

1 Introduction

Integrated Computational Material Engineering (ICME) is "an emerging discipline that aims to integrate computational materials science tools into a holistic system that can accelerate materials development, transform the engineering design optimization process, and unify design and manufacturing.". [23] The notion of ICME arose from the new simulation-based design paradigm that employs a hierarchical multiscale modeling methodology for optimizing load-bearing structures. The methodology integrates material models with structure-property relationships that are observed from experiments at different length scales. ICME emerged because of the recent confluence of smaller desktop computers with enhanced computing power coupled with the development of physically-based material models and associated methods to experimentally validate in-situ rate

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 161–170, 2012.

effects.[21] ICME continues to evolve, its revolutionary vision progressively adapting to assimilate advances in information technology (IT), multiscale simulation technology, and the underlying sciences of engineering (applied mathematics, physics of materials, different length scales, etc.). This paper discusses the strategies for applying the new IT approaches toward the realization of ICME goals.

Cyberinfrastructure is widely recognized for facilitating system-level science through formation of Virtual Organizations (VOs).[29] To construct the VO, the underlying cyberinfrastructure must enable collaborative research by supporting the creation of virtual workgroups and teams, facilitate access to resources, and enhance problem-solving processes.[17] The literature describes many attempts to create effective VOs by capitalizing on dramatic advances in IT. [26] [12] [2] [4] [9] [6] [3] [14] [18]

While it is not possible to create a VO without cyberinfrastructure, the cyberinfrastructure alone is insufficient to establish an effective VO: organizational, human, and social factors are also important aspects of VO creation and operation. For example, the participants must reorient their research methods to work together as a community while still protecting their competitive advantages. Moreover, standardizing patterns of interactions in organizations (e.g., fostering a "corporate culture" for accumulating and sharing the intellectual property)[27]-developing a signature rather than a mere Web site-is critical for the initialization of a successful VO. Depending on the stated goals of a VO, the solutions needed to create and operate the VO may vary. In this paper we present the strategy for the development of a VO aimed toward satisfying the specific needs of ICME, and it is illustrated by actual implementation of crucial parts of the Engineering Virtual Organization for CyberDesign (EVOCD).

The remainder of the paper is organized as follows: in Section 2 we define the goals and functionality of EVOCD and derive the requirements for the supporting cyberinfrastructure. Section 3 describes the functionality of the EVOCD and the software engineering approach adopted to satisfy the requirements of the VO explained in Section 4. Finally, Section 5 summarizes the outcomes of this work.

2 Engineering Virtual Organization for CyberDesign

The primary focus of the ICME vision is establishing a knowledge base accessible to the community-at-large for solving a plethora of disparate issues in material science, applied mechanics, and engineering. This knowledge base requires collection of experimental data describing phenomena at different scales (exploratory experiments, calibration of material models, and validation of models), performing simulations at different scales (atomic, molecular, dislocation, crystal-plasticity, macro-scale FEA), and linking all this information together to determine structure-properties relationships, thereby leading to new concepts and design of new materials. In addition to pushing the edge of material science and solid mechanics by supporting the development and validation of new methods, particularly in the area of multiscale modeling which requires multidisciplinary expertise, the knowledge base is further expected to be used for

engineering design optimization and to support workforce training, including enhancing academic curricula at the graduate level. [21]

It follows that managing the ICME knowledge base directs the principal rationale and objective for establishing a VO. Management entails gathering, developing, integrating, and disseminating experimental data, material models, and computational tools, as well as their use for material and product design. Consequently, the Engineering Virtual Organization for CyberDesign (EVOCD, http://icme.hpc.msstate.edu) is dedicated to the accumulation of the "intellectual capital" pertaining to ICME. It is the organization's capital that attracts community participation in the organization. There are three critical aspects to the process of accumulating capital in order to create a relevant organization: (1) protection of intellectual property, (2) quality assurance of information, and (3) the management of complexity.

In a competitive environment in which materials research and development is performed, protection of the intellectual property is imperative. While the information meant for public consumption must be clearly attributed to its creators, the innovative research may require a restriction of information exchange to only a narrow group of collaborators. The VO must support the former, and enforce the latter. Quality assurance of the information must include its pedigree and then its validation, followed with approval by either a curator or a peer-review process. The management of complexity implies that the information must be easily navigable through intuitive interfaces, yet all complexity of the underlying infrastructure must be hidden from the end user. Furthermore, the information must be understandable to students and directly and efficiently accessible to practitioners.

Notably, many other currently established VOs facilitate widespread collaborative efforts to process huge datasets and require a network of petaflop-range supercomputers to solve specific grand-challenge problems. In this sense, EVOCD is different: it is a cyberspace where the participants can solve their own scientific and engineering problems. On the other hand, EVOCD complements the efforts of nanoHub [6], 3D Material Atlas [1], MatDL [7], NIST Data Gateway [10], and when linked together with these portals, it will become part of the global ICME cyberinfrastructure.

3 Functionality of EVOCD

EVOCD has been developed with the primary goal of accumulating and protecting the intellectual property generated by the participants of the organization. The portal provides powerful passage for accruing and exchanging community knowledge as well as access to repositories of experimental data, material models and computational tools at different length scales, which together exploit the integrative nature of ICME. To achieve this goal, EVOCD is comprised of four primary functional components that are the foundation of the VO: (i) Knowledge Management; (ii) Repository of Codes; (iii) Repository of Data; (iv) Online Calibration Tools.

3.1 Knowledge Management: Wiki

Knowledge management has been achieved by applying an "architecture of participation" as advocated and implemented by Web 2.0 concepts and technologies. Tools like Wiki lead to the creation of a collective (read: peer-reviewed) knowledge database that is always up-to-date with a structure that spontaneously evolves to reflect the current state of the art. Therefore, we have chosen Wiki as the mechanism for community-driven knowledge management.

The Wiki has become the faade for the EVOCD portal to accumulate the knowledge pertaining to ICME. The Wiki captures the knowledge about different classes of materials, material models at various length scales, and design issues, from process and performance models, to optimization under uncertainty. In addition, the Wiki provides direct access to resources, such as data and code repositories.

The intellectual property is protected by configuring the Wiki server [8] to restrict creation and editing of pages to only registered users verified by their email addresses. As a result, all contributions are uniquely attributed to their authors. Following the model introduced by Wikipedia, the quality of contributions is guaranteed by the Web 2.0 process but further monitored by the Wiki editors.

3.2 Repository of Codes

ICME applies computational methods to material science, applied mechanics, and engineering. A significant part of the knowledge is therefore captured as software artifacts from implementing material models, as well as simulation, modeling and optimization codes. Realization of ICME thus critically depends upon providing the capability to gather and disseminate the information about these software components, which is, in turn, an imperative part of the VO's intellectual capital. Consequently, EVOCD serves as the repository of open-source codes contributed by the EVOCD participants. Each code is accompanied with documentation (installation instructions, user manual, theoretical background, and examples). In addition to the open-source material models, the repository provides tutorials and examples for popular commercial or otherwise proprietary codes (such as ABAQUS). The repository of codes complements the knowledge captured in Wiki, enabling the EVOCD user to reproduce the results reported there.

The intellectual property is further protected by restricting access to the actual SVN repository. Only individually-approved contributors have the privilege to offer new revisions. The contributed codes are made available to the general public through a read-only SVN mirror that serves as the back-end for the Web SVN client (open source ViewVC) [13]. All codes downloaded through the ViewVC client are subject to Mississippi State University (MSU) policies and disclaimers. Because of these arguably restrictive policies, many codes listed and documented in the EVOCD repository are available from other locations specified in the repository, typically web sites of their developers or vendors' web sites. This is the

beginning of the "supply chain" envisioned as being the foundation of the global cyberinfrastructure for ICME. The quality of the codes is assured by the fact that they have been used to generate results described in the EVOCD Wiki.

3.3 Repository of Data

Experimental data is another critical component of the intellectual capital captured by EVOCD. At this time, EVOCD focuses on force-displacement, stress-strain, strain-life, and materials characterization data, such as images of microstructure, all of which complement the data repositories offered by other ICME cyberinfrastructure participants. The significance of the data types supported by EVOCD is that they are necessary for the development of Internal State Variable (ISV) material models [22] sed in hierarchical multiscale modeling. The ISV-based models are described in detail in the Wiki pages, and the codes that implement them are available from the repository.

This time, the intellectual property is protected at two levels. At the first level, similarly to the protection of Wiki and repository of codes, only registered users are allowed to contribute. At the second level, the data repository is under access control. To this end, each data request is augmented with SAML-based credentials that are checked against the custom-developed Community Authorization Server (CAS) [19]. This authorization mechanism allows each user to create a group and invite a selected group of users to participate in the group. Only the members of this group are permitted (subject to CAS authorization) to upload data to the group folder. The group moderator (the group creator, or a group member appointed by the group creator) makes the decision to keep the data private, i.e., visible only to the group members, or to make the data "public" by granting read-only access to all users. This group-based access control mechanism is used to exchange restricted-access data, an essential tool for collaborations within EVOCD.

The issue of data quality is addressed in several ways. First, metadata is provided to reveal the pedigree of the data. The information included in a metadata record and pertaining to the data quality is generated automatically from the user session and the mandatory header of the data file. The data is automatically rejected by the system if any critical information is missing (e.g., initial temperature or strain rate for stress-strain data). Most of the publicly available data in the repository have been published in professional journals, thus verified by a peer-review process, and described in the Wiki pages. Non-published data are typically hidden from the general public (group data) and require verification by the group members. Finally, data generated by students are subjected to approval by a curator, most often an academic advisor, and therefore are stored in private group folder. The assurance of the data quality is an example of standardizing the organizational patterns of interactions between the participants defining the organization.

3.4 Online Calibration Tools

The derivation of the material constants from the experimental data to be used by a particular material model is referred to as model calibration, and the capability of model calibration is yet another distinguished feature of EVOCD. Currently, the EVOCD provides three online models for calibration: Plasticity-Damage, Multistage Fatigue, and Thermoplastic; there is also an Image Analysis tool for material characterization [5]. The models are contributed to EVOCD by MSU researchers, available to all users, and their quality scrutinized by the community-at-large. In addition to an intuitive user interface, the tools are functionally integrated with the data repository to facilitate their use; therefore, a selected data set can be seamlessly loaded into the tool, even if it requires data format translation. This defines two important patterns of use possible with EVOCD: (1) the user uploads experimental data, performs model calibration, and saves the material constants in the data repository; (2) the user searches for the constants of a particular model of a particular material and retrieves the constants for further analysis, typically to use them in numerical simulations, such as finite element analysis using ABAQUS or other software.

4 Cyberinfrastructure for EVOCD

The third characteristic of an efficient VO is the management of complexity, which relates to the implementation of the VO and its supporting cyberinfrastructure. One aspect of the management of complexity is the ease of use, which involves, among others features, clarity of presentation, ability to find all relevant information, availability and accessibility of the information for the user's purpose, and hiding from the end-user the intricacies of the underlying infrastructure. Ease of use is realized by the design and implementation of the user interface. Another aspect of complexity management involves maintainability of the VO, including its extensibility, scalability, and most importantly, horizontal integration agility (to avoid messy stow-pipes) to coordinate disparate, autonomous software components into a self-consistent, and perhaps self-healing, unified system. The architecture of EVOCD is shown in Figure 1.

4.1 EVOCD Services

The cyberinfrastructure for EVOCD is a collection of interoperable, autonomous, and platform-independent services to manage and streamline the process of gathering and disseminating knowledge, including computational codes, experimental data, and derived material properties. The Service-Oriented Architecture (SOA) [16] enables the EVOCD portal to hide the details of the heterogeneous platforms and allows integration of services on demand, promoting agility and dynamism to distributed applications in the system. The SOA, which is defined by the Organization for the Advancement of Structured Information Standards (OASIS) as "a paradigm for organizing and utilizing distributed capabilities that may be

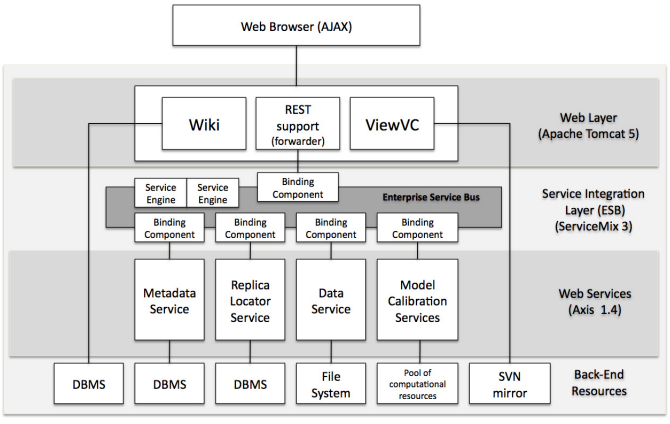


Fig. 1. Multi-tier architecture of EVOCD. The heart of the system is the Enterprise Service Bus that mediates the front-end requests (AJAX) as forwarded to it by the Web layer, and integrates services provided by the back-end resources.

under the control of different ownership domains [25]," empowers the EVOCD cyberinfrastructure to separate functions into distinct services which can be accessed, combined and reused over the network, providing flexibility to adapt to new business prospects and challenges.

EVOCD cyberinfrastructure is comprised of a number of services, notably data, transformation, plotting, computing and authorization services. The data service is an aggregation of three independent "sub-services": metadata, storage, and replica locator services. Each experimental data set in the repository is stored in a file system. The storage service manages the part of the file system designated to store the data sets. When a file is submitted to the storage service, the service determines the location at which the file is to be stored, then returns its URI to the caller. GridFTP is used as the transport mechanism for moving the files to and from storage. The metadata service collects the information about data sets maintained by the storage service. The information is comprised of the file identification (a name assigned by the user, project, material, etc.), the data provenance (owner, date submitted, etc.), tags which enable querying of the metadata repository to find data sets matching particular search criteria, and some additional information necessary to process the data (such as transformation from raw force-displacement measurement to stress-strain relationship). When a new metadata record is created, the service returns its URI so that it can be referred to at a later time. The metadata repository is implemented as a DBMS application. The replica locator provides mappings between the metadata records and data files. There are two significant advantages to this approach. First, the decoupling of the metadata and data storage services permits geographical distribution of the repository to accommodate different storage and access control mechanisms, thus allowing for aggregation of data

services maintained by different organizations. Second, the CAS authorization service can be used to control access to metadata and data separately: the user may be made aware of the existence of the data without being granted automatic access to it. The details of the implementation can be found in [19].

The data services are augmented with independently developed and maintained transformation services, such as format translations (e.g., force-displacement to true stress-strain) and data visualization services, implemented as ESB service engines.

The computing service provides the capability of performing model calibration on selected data sets. The model calibration tools were originally prototyped as the standalone MATLAB applications. However, the implementation of this service required the conversion of an interactive application into a stateless service with actual computations delegated to a back-end computational server. Therefore, to accommodate EVOCD's multi-user environment, a pool of computational engines is scheduled in round-robin fashion. This approach has proven very successful: during a training session, 30 simultaneous users were served by a pool of just four computational engines without any noticeable latencies.

4.2 Service Integration

The final step in developing the cyberinfrastructure for EVOCD is the integration of the disparate autonomous software components (services) into a single unified system capable of processing REST-based requests from the front end. This is achieved by the employment of the Enterprise Service Bus (ESB) [15] [24]. With ESB, requestors and service providers are no longer interacting directly with each other; rather they exchange messages through the bus, and the messages can then be processed by mediations (e.g., message transformation, routing, monitoring). Mediations implement the integration and communication logic, and they are the means by which ESB can ensure that services interconnect successfully. As a result, the ESB acts as the intermediary layer between a portal server and the back-end data sources with which the data portal interacts [28].

EVOCD uses an open-source Apache ServiceMix [11] implementation of ESB. An HTTP-binding component receives REST requests (for security reasons, forwarded by the Apache Tomcat server). A custom router then directs them to the corresponding service provider through the mediation process that involves orchestration of services and message transformations (for the implementation details, see [20]. For example, a simple GET request for a selected data set is processed as a sequence of service invocations: request to the replica locator to convert the data URI to a physical file location followed by GridFTP file transfer. Similarly, to produce a plot visualizing a data set, the GET request parameter must be transformed from the data URI to the location of the local copy of the data retrieved from the repository. Note that this ESB-based mediation process removes all the dependencies of the visualization service on the actual data location. Finally, the model calibration tools require data in a strictly defined format. In the case of format mismatch, the ESB automatically forces format

translation. In general, ESB provides the necessary mechanisms for agile and robust service integration on the one hand, and the bridge between REST and SOA architectures on the other.

5 Conclusions

This paper describes the concept, design, and implementation of Engineering Virtual Organization for CyberDesign (EVOCD) and its supporting cyberinfrastructure. The goal of EVOCD is accumulation of knowledge pertaining to selected aspects of materials science and engineering, and it is expected that EVOCD will become part of the emerging global cyberinfrastructure for the Integrated Computational Material Engineering (ICME). The attractiveness of EVOCD lies in the intellectual capital amassed by the organization, and its future success depends on the capability of protecting the intellectual properties, enforcing the quality of information, and managing the complexity, both for end user and the system developer. The intellectual capital gathered by EVOCD includes experimental data, material models and constants, computational tools and software artifacts, and the knowledge pertaining to multiscale physics-based models for selected properties and processes.

The cyberinfrastructure for EVOCD is a collection of autonomous services, following the Service-Oriented Architecture, with Enterprise Service Bus (ESB) integrating the disparate services into a single unified system to exploit its capability of mediating messages. Furthermore, ESB serves as a bridge between back-end services and AJAX- and REST-based front end services.

EVOCD has been operational for the last 18 months, supporting the research communities involved in the DOE-sponsored Southern Regional Center for the Innovative Lightweight Design, and the Three Nations (Canada, China, and the U.S.) Magnesium Front-End Pilot Project (MFERD). In addition, EVOCD is being extensively used to support the training of engineering graduate students at Mississippi State University. EVOCD is accessible to the general public at http://icme.hpc.msstate.edu.

Acknowledgments. This work has been supported by the U.S. Department of Energy, under contract DE-FC26- 06NT42755 and NSF Grant CBET0742730-08010004.

References

1. Atlas, https://cosmicweb.mse.iastate.edu/wiki/display/home/Materials+Atlas+Home
2. The cancer biomedical informatics grid (cabig), http://cabig.nci.nih.gov
3. Community cyberinfrastructure for advanced microbial ecology research and analysis, https://portal.camera.calit2.net/gridsphere/gridsphere
4. The earth system grid (esg), http://www.earthsystemgrid.org
5. Engineering virtual organization for cyberdesign, http://ccg.hpc.msstate.edu
6. Geon, http://www.geongrid.org/

7. Matdl, http://matdl.org/
8. Mediawiki, http://www.mediawiki.org/wiki/MediaWiki
9. Nanohub, http://nanohub.org
10. Nist data gateway, http://srdata.nist.gov/gateway/
11. Service mix, http://servicemix.apache.org/home.html
12. The southern california earthquake center(scec), http://www.scec.org
13. Viewvc, http://viewvc.org/
14. Datta, A.K., Jackson, V., Nandkumar, R., Zhu, W.: Cyberinfrastructure for chois - a global health initiative for obesity surveillance and control. In: PRAGMA 18, SAN Diego CA (March 2010)
15. Chappel, D.: Enterprise Service Bus: Theory and Practice. O'Reilly Media (2004)
16. Ciganek, A.P., Haines, M.N., Haseman, W.D.: Horizontal and vertical factors influencing the adoption of web services. In: HICSS (2006)
17. Cummings, J., Finholt, T., Foster, I., Kesselman, C., Lawrence, K.A.: Beyond being there: A blueprint for advancing the design, development, and evaluation of virtual organizations. Final report, National Science Foundation (March 2008)
18. Droegemeier, K.: Linked environments for atmospheric discovery (lead): A cyberinfrastructure for mesoscale meteorology research and education. In: 20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology, Seattle, WA (January 2004)
19. Haupt, T., Kalyanasundaram, A., Zhuk, I.: Architecture for a secure distributed repository. In: 7th IEEE/ACM International Conference on Grid Computing, pp. 200–206, No. 170-177 (September 2006)
20. Haupt, T., Kalyanasundaram, A., Zhuk, I.: Using service mashups to implement e-science portals. In: The 2010 IRAST International Congress on Computer Applications and Computer Science (CACS 2010), Singapore (December 2010)
21. Horstemeyer, M.F.: Multiscale modeling: A review. In: Leszczynski, J., Shukla, M.K. (eds.) Practical Aspects of Computational Chemistry, pp. 87–135. Springer, Netherlands (2010)
22. Horstemeyer, M.F., Bammann, D.J.: Historical review of internal state variable theory for inelasticity. International Journal of Plasticity 26(9), 1310–1334 (2010)
23. National Research Council (U.S.):Committee on Integrated Computational Materials Engineering: Integrated computational materials engineering: a transformational discipline for improved competitiveness and national security. National Academies Press (2008)
24. Leymann, F.: Combining web services and the grid: Towards adaptive enterprise applications. In: CAiSE Workshops (2), pp. 9–21 (2005)
25. MacKenzie, C.M., Laskey, K., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0. Architecture 12, 1–31 (2006)
26. National Science Foundation (U.S.), Cyberinfrastructure Council (National Science Foundation): Cyberinfrastructure vision for 21st century discovery (2007)
27. Orlikowski, W.J.W.: The duality of technology: rethinking the concept of technology in organizations. Working Papers 105, Massachusetts Institute of Technology (MIT), Sloan School of Management (2003)
28. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. IEEE Computer 40(11), 38–45 (2007)
29. Stewart, C.A., Almes, G.T., Wheeler, B.C.: Cyberinfrastructure software sustainability and reusability: Report from an nsf-funded workshop. Indiana University, Indiana University (2010)

Dynamic Business Metrics-driven Resource Provisioning in Cloud Environments

Paweł Koperek[1] and Włodzimierz Funika[1,2]

[1] AGH - University, Dept. of Computer Science
al. Mickiewicza 30, 30-059 Kraków, Poland
[2] ACC CYFRONET AGH,
ul. Nawojki 11, 30-950 Kraków, Poland
funika@agh.edu.pl, pkoperek@gmail.com

Abstract. Cloud computing infrastructures are in the spotlight of modern computer science. They offer flexibility and on-demand resource provisioning. The automatic scalability feature enables cloud-based systems to seamlessly adjust to the constantly changing environment of the Internet. Despite their usefulness there is still much space for improvements. In this paper we introduce an approach to automatic infrastructure scaling, based on observation of business-related metrics. An implementation of a tool based on this concept, using the previously developed SAMM system, is also presented. Finally, we discuss evaluation results.

Keywords: cloud computing, automatic scaling, SAMM, Esper.

1 Introduction

Cloud computing platforms [1] become very attractive for a big part of computer software industry. They offer flexibility and pricing options which are very interesting especially from the end user point of view. The service consumer pay only for the actually used resources and need not worry about providing/maintaining them. All the required computing power, memory and disk space can be used on demand [3]. Along with easy allocation and de-allocation, many cloud environments offer the ability to automatically add and remove new resources based on the actual usage. These *automatic scaling* capabilities can provide a great value. With such a tool it is possible to seamlessly deal with peak load situations and to reduce the costs of handling a stream of service requests which form predictable patterns.

Usually the rules behind a scaling mechanism are based on the observation of *generic* metrics, e.g. CPU usage. This approach doesn't require spending additional time to develop customized tools and can be easily applied to many different systems. Nevertheless, it is far from perfect. The decision on what to do with the system is based mainly on low level data, which indicate that the system is already undergoing high load or some resources are not being used. One has also to keep in mind that it always takes some time to execute a particular action. Launching a VM instance, connecting it to load balancing facilities and

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 171–180, 2012.

redirecting requests to a new node may take a while (e.g. a couple of minutes). Therefore the action should be executed at such a moment when it will actually improve the situation, instead of generating an undesired overload of the system.

It is common for monitoring tools to provide not only generic, low level information, but also describe the state of resources with metrics tailored to a specific technology or even a particular instance of the system [2]. In many situations such information indicate how much resources will be required in the near future. For example, if the length of the request queue for computationally intensive tasks is rapidly growing, we may be sure that new virtual machines should be deployed. On the other hand, when request buffers on virtual machines are getting empty, the number of running virtual machines may be reduced.

Based on such observations we developed a novel approach to automatic scaling. We propose to use higher level data, including customized metrics relevant only to a particular system, as decision-making criteria for an auto-scaling mechanism. For example, a resources pool could be extended based on the requests queue length. In this approach we assume that it is far easier for the user to define triggers for dynamic resource provisioning, when they use concepts directly bound to the application to be scaled.

In this paper we present a modified version of Semantic-based Autonomic Monitoring and Management - *SAMM* ([13], [14]) - a system which implements the new paradigm, created as a result of our previous research. The tool enables monitoring a set of customized, high level metrics. They describe the observed system in the context of business objectives (e.g. defined in a Service Level Agreement - *SLA*). Such an insight into the current situation helps administrators to fulfill agreement terms.The latter can be considered by the user as a source for triggering rules. SAMM was designed to independently modify the application's behaviour to prevent it from breaking the contract. Since this level of autonomy isn't always desired, we decided to enhance the system by support for custom rules which trigger specified actions.

The rest of paper is organized as follows: Section 2 presents the already existing approaches in the area of automatic scaling. Next, in Section 3, we provide more details on SAMM's improvements and define (Section 4) an environment which was used to test it. In Section 5 we discuss the obtained results. Finally, Section 6 concludes the paper with a short summary and outlines plans for future work.

2 Related Work

Depending on a user's cloud usage model [4], automatic scaling may be understood in different ways. If the user consumes a ready-to-use software application (*Software as a Service model* [9]) the service provider is the party which is responsible for providing a proper amount of resources. In this case, from the end user perspective automatic scaling is only a feature of the system, which allows to easily satisfy the business needs when they arise. For example, a company

which uses an on-line office software suite may need to provide such software to ten new employees. Instead of buying expensive licenses and installing the software on their workstations, the IT department requests for additional ten accounts in the online service.

In the *Platform-as-a-Service* model ([8], [5]) the situation is similar. The service provider is also responsible for the automatic scaling of application. However, usually the user has to explicitly request for the provisioning of such resources. Providers are able to influence the applications by defining technical constraints for the platform. This way they may ensure that the architecture of the software deployed allows to add and remove some resources dynamically without disrupting normal operation. The algorithm used in decision making may be fine tuned to the underlying hardware and internal resource handling policies of the provider. Usually the user can influence the automatic scaling behavior by setting boundaries of automatic scaling. This prevents from unlimited consumption of resources and therefore from exceeding an assumed budget.

The last model - *Infrastructure-as-a-Service* (e.g. [6]) relies on virtualizing the infrastructure elements like machines and network connections between them. The user has to install and configure everything from scratch and on its top develop their own applications. On the other hand the environment can be customized in many ways, beginning with virtual hardware resources (e.g. CPU power, storage size) and ending with their own server software. Automatic scaling in this model is understood as provisioning on-demand more resources (e.g. virtual machines, virtual storage devices). The user may delegate this task to the service provider ([7]). Adding and removing certain elements is then usually triggered by user-defined rules specifying what action should be taken when a particular threshold is exceeded. These thresholds are limited to a set of metrics predefined by the provider (e.g. CPU usage, storage usage). Many *IaaS* providers also share an API for operations related to managing acquired resources. With such a manner of interaction with infrastructure, the user may create own management tools which can implement custom automatic scaling policies.

In [10] the authors showed that automatic scaling algorithms working with application-specific knowledge, can improve the cost-effectiveness ratio of application deployed in cloud environments. Choosing metrics from a set of traditional system usage indicators as CPU utilisation, disk operation and bandwidth usage can be not helpful enough. The authors decided that the deadline for jobs executed by the system can be used as a key factor for triggering the auto-scaling of the system.

These examples show that currently existing automatic scalability mechanisms can be improved. The presented tools focus on maximizing resource usage, which doesn't have to be the most important factor from the user point of view, e.g. it may be more important to have a system with very short request response time instead of high CPU usage. There are attempts to improve this situation, but there is no generic tool which would be oriented towards easy adaptation to particular systems.

3 Enhancements to SAMM

Our approach to automatic scaling is based on the assumption that for each application it is possible to choose a set of metrics, which can be used to estimate how many resources will be required in the nearest future. On the most basic level it should be sufficient to be able to predict whether more resources will be required, or some of those currently running may be stopped. This way the user can determine thresholds related to triggering some actions which influence the system in a desired way.The set of metrics is tightly coupled with the application's specifics. Even those metrics with the same names may have different semantics if used in different contexts. To handle so much different information, a data representation capable of describing all the used concepts is required. Additionally it should be possible to easily extend the monitoring system with support for new data acquisition techniques. Measurements may have to be gathered with use of several technologies.

Therefore, we have decided to use the result of our previous work on automatic scaling - SAMM ([13], [14]), as a starting point. It uses ontologies to describe resources and available metrics so it is possible to describe very different system architectures. Owing to its module-based architecture based on OSGi bundles and services, adding support for new technologies or replacing the existing components doesn't require much effort. To meet the requirement of being able to define rules in a convenient way, we came to a new decision-making module. For this purpose we exploited the Esper ([15]) event processing engine. The internal architecture of the enhanced SAMM is depicted in Fig. 1.

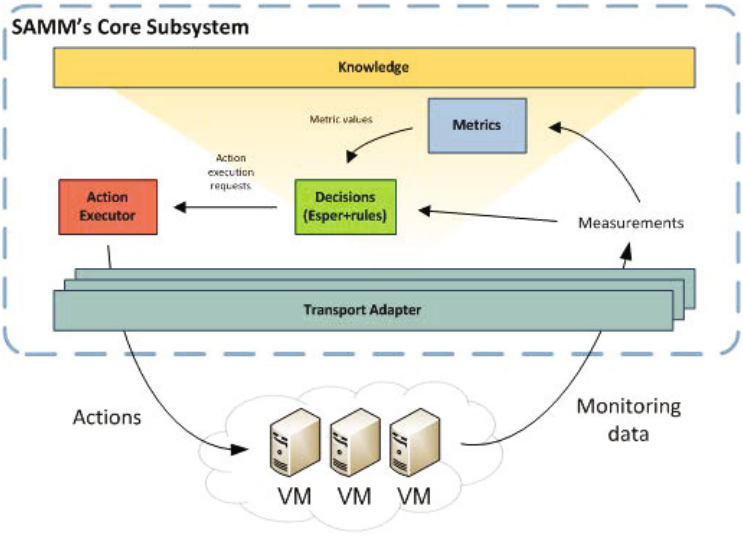


Fig. 1. SAMM's architecture after changes

The flow of measurement activities is as follows:

1. Measurements are collected by the *Transport Adapters* which are a an abstraction layer over different data collection technologies, e.g. there are separate adapters for Java Management Extensions (*JMX*, [18]) and Eucalyptus ([16]). They translate internal SAMM measurement requests to a specific technology language, e.g. Java method invocations.
2. The *Metrics* module processes measurements according to metric formulas into metrics values and sends them further to the *Decisions* module.
3. Values coming from *Transport Adapters* and *Metrics* modules are processed by the *Decisions* module. Measurement and metrics values are processed by Esper as events. They are filtered with the rules provided by the user.
4. Whenever the *Decisions* module detects exceeding a threshold, a request to execute an action is sent to *Action Executor*. This component tries to modify the infrastructure via a specific *Transport Adapter*.

The rules can trigger the execution of certain actions. An action may simply be a Java code bound to a particular communication protocol. Due to Esper conditions are very flexible (aggregation, filtering, etc.) the only limitation being the flexibility of the Esper query language.

4 Evaluation of the Approach

To evaluate our approach to automatic resources provisioning we applied a scaling policy based on business-metrics to a sample application - a simple service which provides numerical integration (please see in Fig. 2). To easily scale the number of nodes used for computation, the *Master - Slave* model was used. The *master* node dispatches the requests (functions to be integrated) and one or more *slave* nodes which perform numerical integration [1].

The *Master* node has three components:

- **Slave Dispatcher** - the main component of *Master* node. It handles the queue of incoming integration requests. If any request is present in the queue, **Slave Dispatcher** removes it from the queue and sends a numerical integration request to one of the registered slaves. A *slave* is chosen based on the following algorithm:
  1. Retrieve a proxy for another slave from **Slave Resolver**
  2. If the slave is capable of handling a next request, send it to this node, else go to point 1.
  3. If there are no slaves capable of handling the request - wait (e.g., 5 seconds) and start from the beginning.

  The slave node is capable of handling a next request, if it does not overflow the buffer of numerical integration requests. The size of the buffer was set up to 25 requests.

[1] We do not focus on the details of the numeric integration algorithm.

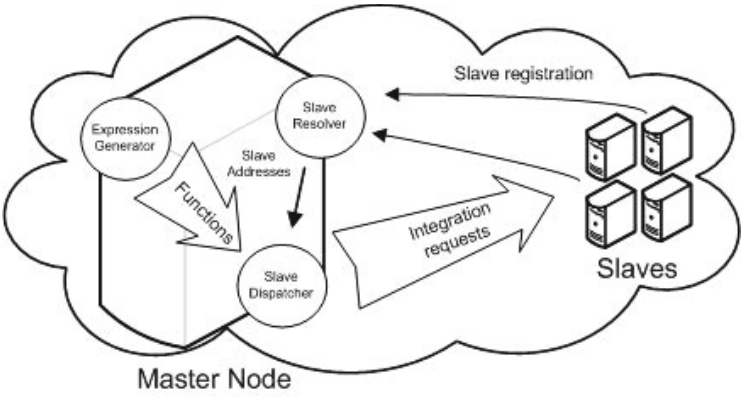


Fig. 2. Test application architecture

- **Slave Resolver** - the component in which slave nodes register. The information about the addresses of nodes are later shared with **Slave Dispatcher**.
- **Expression Generator** - generates the functions to be numerically integrated. The functions can be generated infinitely or read from a file in chunks of a specified size.

4.1 Test Environment

All the tests were carried out on the FutureGrid project environment [17], on the **India** Eucalyptus ([16]) cluster. This cloud environment provides the following virtual machine types:

- `m1.small` - 1 CPU, 512 MB of RAM, 5 GB of storage space
- `c1.medium` - 1 CPU, 1024 MB of RAM, 5 GB of storage space
- `m1.large` - 2 CPUs, 6000 MB of RAM, 10 GB of storage space
- `m1.xlarge` - 2 CPUs, 12000 MB of RAM, 10 GB of storage space
- `c1.xlarge` - 8 CPUs, 20000 MB of RAM, 10 GB of storage space

The cluster contains 50 nodes and provides up to 8 small instances per node. Since *slave* nodes didn't use much storage space and memory, we decided to use `m1.small` instances to run them. *Master* node's application had higher memory requirements so it had to use the `c1.medium` instance in this case. The advantage of using the smaller instances was a possibility to control the number of CPUs involved in the computation at a fine-grained level.

4.2 Test Cases

We evaluated our approach by comparing two strategies of automatic scaling. The first one was based on a generic metric - CPU usage. The second one used a business metric - the average time spent by computation requests while waiting for processing in *Slave Dispatcher's* queue.

The rules used in the first approach are:

- Start another *slave* node virtual machine, when the average CPU usage of *slave* nodes from the last 300 seconds was higher than 90%
- Stop a random *slave* node virtual machine, when the average CPU usage of *slave* nodes from the last 300 seconds was less than 50%.

The rules exploited in the second approach are as follows:

- Start another *slave* node virtual machine, when the average wait time of request from the last 300 seconds was higher than 35 seconds
- Stop a random *slave* node virtual machine, when the average wait time of request from the last 300 seconds was less than 10 seconds.

One virtual machine had to be running all the time and at most ten instances could get started automatically. Such limitations have to be considered before starting work with automatic scaling algorithms. Otherwise all available resources could get consumed or the application could get shut down.

One has to bear in mind that these parameters were tuned up specifically to the infrastructure on which we carried out the tests and its current load. The FutureGrid Eucalyptus **India** cluster was used in parallel by other users, thus e.g. the start time of virtual machines varied over time.

During the evaluation two test scenarios were considered. Both of them consisted of 360 steps. Each step consisted of adding some functions to the *Slave Dispatcher's* incoming requests queue by *Expression Generator* and waiting 60 seconds. The first workload is used to verify if SAMM is able to properly adjust the amount of resources to handle a constant stream of requests. The second one shows that the system can optimize the amount of allocated resources to meet certain performance goals. To ensure that two auto-scaling strategies will handle the same situation, the functions to be processed (randomly generated polynomials) were generated and stored into files before the actual test. During the test run *Expression Generator* read functions from the file and passed them further for processing.

The number of requests added to the queue was equal to 100 in each step of *static workload* while being equal to

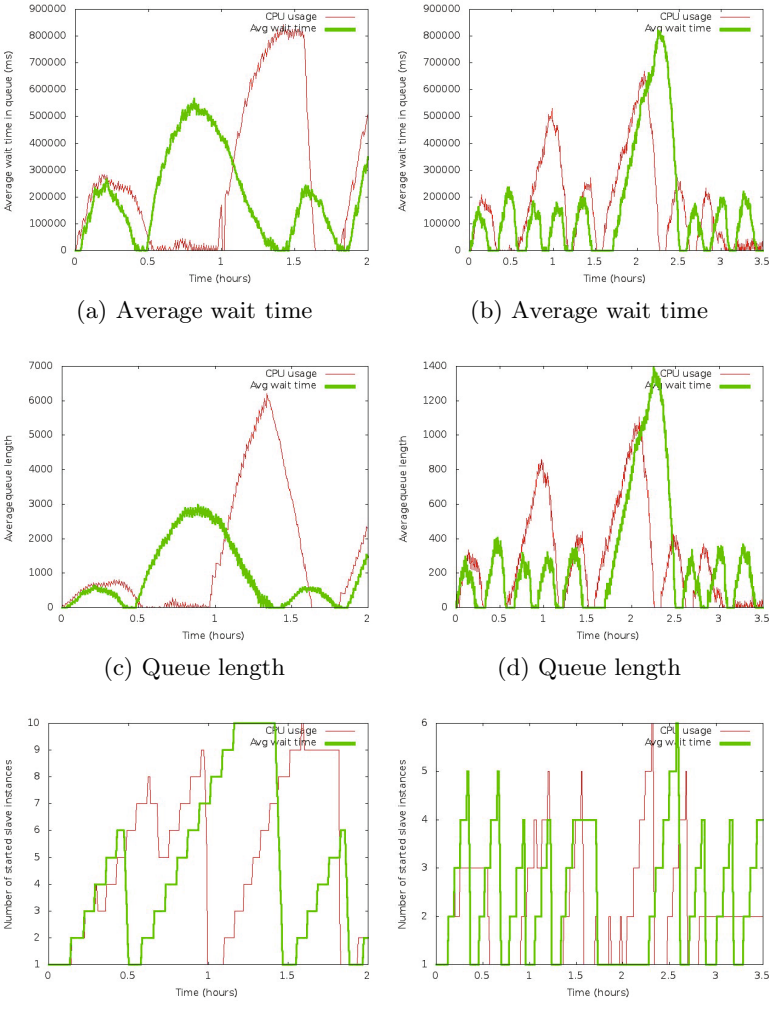
$$ExprNum(n) = 100 + 5 * (n \ mod \ 80)$$

(where `n` is the number of iteration) of each step of *dynamic workload*.

5 Results

To compare the automatic scaling when using the selected strategies, we investigate the average wait time, *Slave Dispatcher's* input queue length and the number of running instances for both dynamic and static workloads (Fig. 3). Such metrics can be used in SLA for describing the features which are crucial from the business viewpoint.

For the dynamic workload, during the first half of an hour, SAMM launches too many slave nodes. The number of integration requests is still growing, therefore machines are kept being busy.



(a) Average wait time (b) Average wait time

(c) Queue length (d) Queue length

(e) Number of running instances (f) Number of running instances

Fig. 3. Test results for dynamic *(a,c,e)* vs. static *(b,d,f)* workload execution using two strategies (*CPU usage* and *Avg wait time*)

In the first approach (generic metrics) it triggers the allocation of more and more resources until there are too much slaves. When the queue gets completely empty, CPU usage drops and SAMM terminates the unused virtual machines. However, after some minutes, the queue gets refilled with new requests and now more resources have to be provisioned to handle the load. After 90 minutes (when the workload rapidly drops), the number of requests in the queue is still large. In the second approach (business metrics) the rapid drop of the average wait time also indicates that some resources should get released. However in

this case, SAMM can recognize early that the workload still is increasing, and provides necessary resources. Once the workload has dropped, the number of running virtual machines is declining as well.

For the static workload the differences between the first and second strategy are significantly smaller. Both strategies had a problem with setting up a proper number of virtual machines that led to the accumulation of a big number of requests.

Table 1 presents the average values of the metrics.

Table 1. Average metric value for AvgWaitTime and CPU strategies

	Dynamic workload		Static workload	
Metric	AvgWaitTime	CPU	AvgWaitTime	CPU
Average instances number	4.53	4.96	2.31	2.12
Average wait time (ms)	203987.08	266362.25	161727.80	167754.66
Average queue length	934.06	1392.72	273.42	269.65

6 Conclusions and Future Work

The results on the use of two VM allocation policies presented in the paper show that using the average wait time metric may have a positive impact on the system from the business point of view. End users are mostly interested in making the time required to wait as short as possible. By improving this factor, the potential business value of the sample service shows an increase. Instead of focusing on the CPU usage, the resource management algorithm was controlled by a far more significant metric, what is important, from the business value perspective.

The actual improvement highly depends on workload. For the first test scenario, the system shortened the time spent on waiting by 62 seconds. However, the second case showed that there are situations in which a strategy based on generic metrics can be as good as the one based on business metrics.

Automatic scalability features of clouds are a response for very quickly changing environments. Adding resources on-demand can enable a running system to smoothly handle high request loads and serve a bigger number of users. On the other hand when the demand for certain resources goes down, the unused capacity is disabled, what makes the operational costs lower.

The possibility to make the automatic scaling facilities more aware of business rules makes them even more useful, especially in private cloud systems. With better insight into what elements are running inside the system, scaling rules can be fine-tuned to particular hardware and software setup. A proper ratio of the resources allocated to the most crucial applications to the computing power spent on low priority tasks can be automatically maintained.

Our research in automatic scaling area is ongoing. The nearest plans include development of a user-friendly web interface for SAMM which would facilitate

using the service. Another goal is to add support for other cloud stacks, e.g. Open Stack [12] or Open Nebula [11] thus making SAMM interoperable in a heterogeneous environment.

Acknowledgments. The research presented in this paper has been partially supported by the European Union within the European Regional Development Fund program no. POIG.02.03.00-00-007/08-00 as part of the PL-Grid Project (www.plgrid.pl) The experiments were carried out using the FutureGrid project resources [17].

References

1. Buyya, R., et al.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. FGCS 25(6), 599–616 (2009)
2. Bubak, M., Funika, W., Wismueller, R., Metel, P., Orlowski, R.: Monitoring of Distributed Java Applications. FGCS 19(5), 651–663 (2003)
3. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications 1, 7–18 (2010)
4. Rimal, B.P., Choi, E., Lumb, I.: A Taxonomy and Survey of Cloud Computing Systems. In: 5th Int'l Joint Conf. on INC, IMS and IDC, NCM 2009, pp. 44–51 (2009)
5. Amazon Web Services LLC, http://aws.amazon.com/elasticbeanstalk (access: May 09, 2011)
6. Amazon Web Services LLC, http://aws.amazon.com/ec2 (accessed: April 26, 2011)
7. Amazon Web Services LLC, http://aws.amazon.com/autoscaling (accessed: May 13, 2011)
8. Google Inc, App Engine project website, http://code.google.com/appengine (accessed: April 26, 2011)
9. Google Inc., Google Apps website, http://www.google.com/apps (accessed: April 26, 2011)
10. Mao, M., Li, J., Humphrey, M.: Cloud Auto-scaling with Deadline and Budget Constraints. In: 11th IEEE/ACM Int'l Conf. on Grid Computing (Grid 2010), pp. 41–48 (2010)
11. Open Nebula project website, http://opennebula.org (accessed: May 13, 2011)
12. Open Stack project website, http://www.openstack.org (accessed: May 13, 2011)
13. Funika, W., Kupisz, M., Koperek, P.: Towards autonomic semantic-based management of distributed applications. In: Computer Science Annual of AGH-UST, vol. 11, pp. 51–63. AGH Press, Krakow (2010) ISSN 1508-2806
14. SAMM Project, http://code.google.com/p/sammanager (accessed: May 13, 2011)
15. Esper Project, http://esper.codehaus.org (accessed: May 10, 2011)
16. Eucalyptus Systems, Inc., http://www.eucalyptus.com (accessed: May 13, 2011)
17. FutureGrid Project, https://portal.futuregrid.org (accessed: May 13, 2011)
18. Oracle Corporation, http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html (access: May 14, 2011)

Stochastic Control of the Scalable High Performance Distributed Computations

Zdzislaw Onderka

Dept. of Geoinformatics and Applied Computer Science,
AGH, Univ. of Science and Technology, Krakow, Poland
zonderka@agh.edu.pl, onderka@ii.uj.edu.pl

Abstract. The paper presents model desribing the dynamics of the background load for the computers in the network. For that model two problems of the task allocation in the network are formulated: the problem of the stochastic control based on the background load model and the stochastic control based on the Markov Decision Process Theory. The *open-loop* and *closed-loop* control algorithms based on the stochastic forecast and Markov Decision Process Theory respectively are presented.

Keywords: stochastic control, distributed computation, optimal task distribution, Markov Decision Process.

1 The Manager-Worker Application and Heterogeneous Network Model

The *heterogeneous computer network and distributed application* is a tuple $< H, A, F, G_A >$ [3] [7], where:

- $H = \{M_0, \ldots, M_m\}$ is a set of $m+1$ distinct machines which may communicate with any other one, may have a different architecture and for each machine its performance parameters may vary in time. We assume that the computer network is *nondecomposable* [1].
- A is a *manager-worker* application which is composed of sequential part t_0 (*manager task*) and distributed part consisting of N identical tasks (*workers*) $V_D = \{t_1, \ldots, t_N\}$.
- F is a mapping $F: V \to H$ which in case of the dynamic tasks allocation is defined as a sequence of partial mappings, $\{F_n\}_{n=0,1,\ldots}$, i.e.

$$\bigcup_n F_n^{-1}(M_i) = F^{-1}(M_i) \quad \forall M_i \in H \qquad F^{-1}(M_0) = t_0 \tag{1}$$

- $G_A = (V, E)$, is a *scheduled task graph* for the application A [1] defined by F;

2 The Estimation of the State of Background Workload

The considered network H is composed of the computers of which resources are shared between several users. Therefore, processes may appear randomly on

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 181–190, 2012.

each machine $M \in H$, so the background load of M changes during the time. For that reason the background load of the machine is allowed to be estimated in the stochastic way.

Let $load_j \in \mathcal{R}_+$ be the additional background load for the machine $M_j \in H$. Each value $load_j$ corresdponds to the *execution slowing down parameter* $\eta_j \in \mathcal{R}_+$ [4] [7] i.e. $\overline{T}_t^j = (1+\eta_j) \cdot T_t^j$, where $\overline{T}_t^j$ and T_t^j are the pattern task execution times with and without additional background load respectively. So, further we will consider the bijection $\psi_j : \mathcal{R}_+ \ni load_j \to \eta_j \in \mathcal{R}_+$.

Let's consider Θ_j and Γ_j two ordered sets of thresholds ($K_j \in \mathcal{N}$):

$$\Theta_j = \{\theta_j(i) \in \mathcal{R}_+,\ i = 0, \ldots, K_j : \forall i = 0, \ldots, K_j - 1 \ \ \theta_j(i) < \theta_j(i+1)\} \quad (2)$$

$$\Gamma_j = \{\eta_j(i) \in \mathcal{R}_+,\ i = 0, \ldots, K_j : \forall i = 0, \ldots, K_j - 1 \ \ \eta_j(i) < \eta_j(i+1)\} \quad (3)$$

First of them refers to the physical measured performance parameters (like $load_j$) and the second one to the slowing down parameters, and such that $\forall M_j \in H \ \ \forall i = 0, \ldots, K_j \ \ \psi_j(\theta_j(i)) = \eta_j(i)$. So, $\forall M_j$ we obtained a set $S_j = \{0, \ldots, K_j\}$ which will be called the *set of states of the background load*.

Let's define the mapping $\Psi_j : \mathcal{R}_+ \to S_j$ as:

$$\Psi_j(\eta_j) = \begin{cases} 0 & for \ \ \eta_j(0) \le \eta_j < \eta_j(1), \\ 1 & for \ \ \eta_j(1) \le \eta_j < \eta_j(2), \\ \ldots & \ldots \\ K_j \ dla & \eta_j(K_j) \le \eta_j < \infty \end{cases} \quad (4)$$

where $\forall j \ \ \eta_j = \psi_j(load_j)$.

Now, the machine $M_j \in H$ is in the state $i \in \{0, \ldots, K_j\}$ of the background load $iff \ \ \Psi_j \circ \psi_j(load_j) = i$. Moreover, $\forall M_j$ let define bijection $\varphi_1^j : S_j \ni i \to \eta_j(i) \in \Gamma_j$.

3 The Model of Background Workload for the Computer Network

We assume a tuple $<\wp, \{X_n\}_{n=0,1\ldots}, \mathcal{P}, \Gamma, \Theta, S, \varphi_1>$ as the *model of background load* for the computer network H where

- $\wp$ is a tuple $<\wp_1, \ldots, \wp_m>$ where $\forall j \ \ \wp_j = (\Omega, \Im, P^j)$ is a probability space where Ω is a set of events that M admits some average value of the execution slowing down parameter in the time interval Δ_n for some n, identical for each M_j. In consequence each evant corresponds with an unique state from S_j.
- $\{X_n\}_{n=0,1\ldots}$ is a tuple $< \{X_n^1\}_{n=0,1\ldots}, \ldots, \{X_n^m\}_{n=0,1\ldots} >$ of nonstationary discrete stochastic Markov chains [6] which describe dynamic behavior of state of background load for each M_j during the time. The dynamics is given by:

$$P^k(X_{n+1}^k = j | X_n^k = i) = P^k(X_{n+1}^k = j | X_0^k = i_0, \ldots, X_{n-1}^k = i_{n-1}, X_n^k = i) \quad (5)$$

  Each $X_n^j : \Omega \to S$ corresponds to the time interval $\Delta_n, n = 0, 1, \ldots$.

- $\mathcal{P}$ is a tuple $<\mathcal{P}^1,\ldots,\mathcal{P}^m>$ where $\mathcal{P}^k=\{P_n^k\}_{n=0,1\ldots}$ is a sequence of Markov transition matrices. The $p_{ij}^k(n)=P_n^k(X_{n+1}^k=j|X_n^k=i)$ is a probability that process will be in state j of background load at the step $n+1$ provided that it is in state i at step n for the machine M_k. If $\Pi(n)=\left(\Pi^1(n),\ldots,\Pi^m(n)\right)$ denotes the vector of the state probability distributions at the step n for the network H and $\Pi^j(\mu)=\beta_\mu=(\beta_\mu^1,\ldots,\beta_\mu^m)$ is the initial distribution, then the consecutive distributions may be evaluated according to the formula [2, 6]: $\Pi^j(n)=\Pi^j(n-1)\cdot P_n^j,\quad n>\mu$.
- Γ is a tuple $<\Gamma_1,\ldots,\Gamma_m>$ where Γ_j is the following set of tresholds for the slowing down parameters:

$$\Gamma_j=\{\eta_j(i)\in\mathcal{R}_+,\ i=0,\ldots,K,:\forall i=0,\ldots,K-1\ \ \eta_j(i)<\eta_j(i+1)\}\quad(6)$$

- Θ is a tuple $<\Theta_1,\ldots,\Theta_m>$ where Θ_j is the following set of tresholds for the physical measured performance parameters for the machine M_j:

$$\Theta_j=\{\theta_j(i)\in\mathcal{R}_+,\ i=0,\ldots,K,:\forall i=0,\ldots,K-1\ \ \theta_j(i)<\theta_j(i+1)\}\quad(7)$$

  Moreover, $\forall j=1,\ldots,m\ \ \forall i=0,1,\ldots,K\quad \psi_j:\Theta_j\ni\theta_j(i)\rightarrow\eta_j(i)\in\Gamma_j$;
- S is a tuple $<S_1,\ldots,S_m>$ where S_j is the set of states of background load for the machine $M_j\in H$ and $card(S_j)=card(\Gamma_j)\ \forall j$.
- φ_1 is a tuple $<\varphi_1^1,\ldots,\varphi_1^m>$, $\forall j=1,\ldots,m$ φ_1^j is a mapping $\varphi_1^j:S_j\rightarrow\Gamma_j$.

Having the stochastic processes $\{X_n^j\}_{n=0,1,\ldots}$ which describes dynamic behavior of the background load of M_j and bijections φ_1 and φ_2 $(\varphi_2=<\varphi_2^1,\ldots,\varphi_2^m>$ where $\varphi_2^j:\Gamma_j\rightarrow\mathcal{R}_+$, and $\varphi_2^j(\eta_j)=(1+\eta_j)\cdot T_t^j)$, then we may define the new stochastic processes $\{\overline{T}_t^j(n)\}_{n=0,1,\ldots}$ describing the dynamic behavior of the times required to execute the single task $t\in V_D$ on the machine M_j with the background load. Moreover, $\forall j\in\{1,\ldots,m\},\ n=0,1,\ldots$

$$\overline{T}_t^j(n)=\varphi_2^j\circ\varphi_1^j(X_n^j))=(1+\eta_j(X_n^j))\cdot T_t^j\quad(8)$$

4 The Stochastic Control Based on Background Load Model

The *control policy* u is the sequence $\{F_n\}_{n=\mu,\mu+1,\ldots}$ where the subscript $\mu\geq 0$ denotes the starting time epoch τ_μ (the time step of the nonstationary stochastic process describing dynamic behavior of the background load) of the realized policy.

Let U_μ be the set of *admissible policies* i.e. policies which accomplish defined limitations. The policy $u\in U_\mu$ is an *admissible policy* if $\exists q\in(0,1]$, $\forall j=1,\ldots,m\ \ \forall n=\mu,\mu+1,\ldots$

$$q\leq P_n^j\left(\left(\overline{T}_t^j(n)\cdot card(F_n^{-1}(M_j))\right)<\Delta_n\right)\leq 1\quad(9)$$

where $P(\cdot)$ denotes the probability and $\Delta_n=[\tau_n,\tau_{n+1})$ denotes the time interval.

Let $C_n(\beta_\mu, s, u)$ denotes the *cost function for the step* $n=\mu, \mu+1, \ldots$ of the realized policy $u \in U_\mu$ (consequently, that is the cost of the function F_n applied in the period $[\tau_n, \tau_{n+1}),\ n = \mu, \mu+1, \ldots)$ defined for the initial distribution $\beta_\mu = (\beta_\mu^1, \ldots, \beta_\mu^m)$ of the state of background load in the time epoch τ_μ and achieved state $s = (s^1, \ldots, s^m),\ s^j \in S_j$ in τ_n. It is defined as:

$$C_n(\beta_\mu, s, u) = \max_{M_j \in H} (c^j(n, s^j, u)) \tag{10}$$

where $c^j(n, s^j, u)$ is an *immediate cost function* for machine $M_j \in H$ that is in the state $s^j \in S_j$. Also, let'sassume that $\forall u \in U_\mu$, $\forall \mu$ and $\tau_n \geq \tau_\mu$, and for each initial distribution β_μ^j and each state in the time epoch τ_n for M_j there exists an expected value $E_{\beta_\mu^j}\left(c^j(n, s^j, u)\right)$ [2] [6].

Let $C^Z(\beta_\mu, u)$ be the *finite horizon cost for the network H* related to the policy $u \in U_\mu$ provided in the time period $[\mu, Z], (Z < \infty)$. It is defined as:

$$C^Z(\beta_\mu, u) = E_{\beta_\mu}\left(\sum_{n=\mu}^{\mu+Z-1} C_n(\beta_\mu, s, u)\right) \tag{11}$$

We define the **control problem** as:

Find a policy $u \in U_\mu$ *that minimizes* $C^Z(\beta_\mu, u)$ *over the whole* U_μ .

5 The Stochastic Control Based on MDP

The following control problem is based on the theory of Markov Decision Process (MDP) [5] [6]. Let $\mathcal{T} = \{\tau_\mu, \ldots, \tau_{\mu+Z-1}\}$ be a set of decision epochs correspodning to the beginning of the period $\Delta_n,\ n = \mu, \mu+1 \ldots$ and $Z \in \mathcal{N},\ Z < \infty$. We assume that the last decision is made at $\tau_{\mu+Z-1}$.

In each decision epoch the *decision agent* receives the state $s=(s^1, \ldots, s^m) \in S$ of network H in which may choose an action $a = (a^1, \ldots, a^m) \in A_s = A_{s^1}^1 \times \ldots \times A_{s^m}^m$, where $A_{s^k}^k$ is the discrete set of actions which may be chosen if the state of machine M_k is s^k. Moreover, let's assume that the actions are chosen in deterministic way.

As a result of choosing the action $a^k \in A_{s^k}^k$ in the state s^k at the decision epoch τ_i is an *immediate cost function* $c^k(i, s^k, u)$ $(u \in U_\mu)$ and the state of machine M_k in the decision epoch τ_{i+1} is determined by the probability distribution $p_{\tau_i}^k(\cdot|s^k, a^k)$.

A *decision rule* describes a procedure for action selection in each state and decision epoch τ_i. It is defined as a function $d_\tau : S \ni s \to d_\tau(s) \in A_s$, where if $s = (s^1, \ldots, s^m)$ then $d_\tau(s) = (d_\tau^1(s^1), \ldots, d_\tau^m(s^m))$.

The *control policy* is defined as the sequence $u = (d_0, d_1, \ldots, d_{Z-1})$. We will use the finite horizon Markov deterministic policies [5] [6] in order to control the execution of the distributed application. It means that each decision rule depends on previous state of the machines whole workload and selected action in this state, and each action is chosen with certainty. The whole workload for

the machine will be understood as the common load of background and load derived from the execution of the task belonging to an application A.

The *admissible policy* u is defined by the formula (9) where if $s_n = (s_n^1, \ldots, s_n^m)$ and $a_n = (a_n^1, \ldots, a_n^m)$ than

$$\{d_n(s_n)\}_{n=0,1,\ldots,Z-1} = \{a_n\}_{n=\mu,\mu+1,\ldots} = \{F_n\}_{n=\mu,\mu+1,\ldots} \tag{12}$$

In the finite horizon Markov decision processes we define the sample space Ω^H for the network H as $\Omega^H = \{S \times \mathcal{A}\}^{Z-1} \times S$ so the elementary event $\omega^H \in \Omega^H$ is a sequence of states and actions, that is: $\omega^H = (s_0, a_0, s_1, a_1, \ldots, a_{Z-1}, s_Z)$ $\forall n \;\; s_n = (s_n^1, \ldots, s_n^m),\; a_n = (a_n^1, \ldots, a_n^m)$.

Now, let's define for each machine M_k the sequence of random variables $\overline{X}_i^k(\omega)$, $\omega \in \Omega$, $i = 0, 1, \ldots$ as $\overline{X}_i^k : \Omega \to S^k$ and $\overline{X}_i^k(\omega) = s_i^k$ where s_i^k is the state of the common load of background and load derived from the execution of the task belonging to an application A (the whole workload for the machine). And additionaly let's define the sequence of the random variables $Y_i^k(\omega)$, which $\forall i$ admits values $Y_i^k(\omega) = a_i^k \in A_{s_k}^k$. If we denote by h_i^k the history for the machine $M_k \in H$ i.e.

$$h_i^k = (\overline{X}_0^k = s_0^k,\; Y_0^k = a_0^k, \ldots, Y_{i-1}^k = a_{i-1}^k,\; \overline{X}_i^k = s_i^k) \tag{13}$$

then the dynamic behavior of the state of the whole workload of machine M_k is given by:

$$\begin{aligned} &P_u^k\left(\overline{X}_{n+1}^k = j | h_n^k = (h_{n-1}^k, a_{n-1}^k, i), Y_n^k = a_n^k\right) = \\ &P_u^k\left(\overline{X}_{n+1}^k = j | \overline{X}_n^k = i, Y_n^k = a_n^k\right) = p_{\tau_n}^k(j|i, a_n^k) \end{aligned} \tag{14}$$

i.e. the next state of the whole bachground depends only on the current state and chosen action in this state. The state probabilities $\overline{\Pi}^j(\mu+i+1)$ for random variables $\overline{X}_{\mu+i+1}^k$ at the decision epoch $\tau_{\mu+i+1}$ can be evaluated by:

$$\overline{\Pi}^k(\mu + i + 1) = \sum_{j \in S_k} p_{\tau_{\mu+i}}^k(j \mid s_j^k, a_j^k) \cdot \overline{\Pi}^k(\mu + i)) \tag{15}$$

where $\overline{\Pi}^k(\mu + i)$ is the probability distribution at the decision epoch $\tau_{\mu+i}$.

The computation of the probability distribution $\overline{\Pi}^k(\mu + i)$ is based on the probability distribution of the state of background load. Let's sign $r_i = \eta(i+1) - \eta(i)$ and $\alpha_i = \delta / r_i$ where δ is the load derived from the task of an application A. Let's assume that the load δ is less then r_i, then the evaluation of the probability distribution $\overline{\Pi}^k(\mu + i)$ is as follows (in matrix form) ([4]):

$$\left(\overline{\Pi}^k(n)\right)^T = B \cdot \left(\Pi^k(n)\right)^T = B \cdot \left(P^k(n) \cdot \Pi^k(n-1)\right)^T \tag{16}$$

The superscript T denotes the matrix transposition and matrix B has the following form:

$$\begin{bmatrix} 1-\alpha_1 & 0 & 0 & 0 & \cdots \\ \alpha_1 & 1-\alpha_2 & 0 & 0 & \cdots \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \cdots \\ \cdots & & & & \cdots \\ 0 & \ldots & & \alpha_{K-1} & 1 \end{bmatrix} \tag{17}$$

The *finite horizon cost for network H* for the deterministic Markov policy $u \in U_\mu$ and for the finite horizon Z is defined as:

$$C^Z(\beta_\mu, u) = E^u_{\beta_\mu}\left(\sum_{n=\mu}^{\mu+Z-1} C_n(n, \overline{X}_n, d_n(h_n)\right) \tag{18}$$

where $h_n = (h_{n-1}, a_{n-1}, s)$, and $\beta_\mu = (\beta^1_\mu, \ldots, \beta^m_\mu)$.

Now, the control problem defined in Section 4 lies in finding the policy $u \in U_\mu$ that minimizes the defined above finite coast function (18) over the U_μ.

The existance of the optimal Markov deterministic policy is guaranteed by the following theorem [6]:

Theorem 1. *Assume S is finite or countable. Then if*

1. *$\forall s$ the set of actions A_s is finite, or*
2. *A_s is compact, than the expected cost function $C_n(\beta_\mu, s, a)$ for the step n of the realized policy is continuous in point a $\forall s \in S$, and $\exists C < \infty$ for which $C_n(\beta_\mu, s, a) \leq C$ $\forall a \in A_s$, $s \in S$, and $p^k_n(j \mid s^k_n, a^k_n)$ is continuous in a^k_n $\forall j \in S_k$ and $s^k_n \in S^k$ and $n = \mu, \ldots, Z-1$*

then there exists a deterministic Markovian policy which is optimal.

6 The Policies of Tasks Distributions

In thise paper will be considered policies which are leading to the fastest execution of an application A. They fall generally in two groups: the group of deterministic policies and the group of stochastic policies based on the described Markov models.

The following policies from the first group: *single task*, *multiple task*, *dynamic single distribution*, *stationary* are described in details in [3] [4].

In order to execute one of the stochastic policies, we introduce the following classes of agents: *state agents* - that monitors the load during the time and prepares the state forecast for each machine in H; *decision making agents*, that are involved in establishing task allocation policy based on actual forecast of the average state of each machine in H (or forecasted computation time of task) [7]. Below there are presented two algorithms belonging to the group of stochastic policies: first one based on the background load model (see Sect. 3) (*open loop control*) and the second one based on the MDP model (see Sect. 4) (*closed loop control*) [4].

6.1 The Open Loop Control

The following algorithm is based on the average state of the background load forecast generated only once in the starting time epoch τ_μ for each machine in H. More precisely, this policy rely on the computation of the distributions $\mathit{\Pi}_i^k(n)$ having the initial distributions of the state of background load β_μ^k for each machine $M_k \in H$ recursively for each time epoch according to the formula:

$$\mathit{\Pi}_i^k(n) = \sum_{j \in S} p_{ji}^k(n-1) \cdot \mathit{\Pi}_j^k(n-1), \quad \tau_n > \tau_\mu \tag{19}$$

Now, having the vector of the state probabilities for each time epoch we may compute the expected execution times of one task on each machine in each time step which leads to the computation of *power coefficients* λ_j for all machines M_j as well as *power coefficient* $\mathit{\Lambda}$ for the network H defined as: $\lambda_j = \frac{1}{\overline{T}_t^j}$, $\mathit{\Lambda} = \sum_{j=1}^m \lambda_j$.

The consecutive decision rules (or functions F_n) allocates the subsets of tasks $F^{-1}(M_k)$ for which cardinal numbers are proportional to the machines expected *power coefficients* for the $n = \mu, \mu+1, \ldots, \mu+Z-1$.

Algorithm 1

Step 1: $n = \mu$; $V(n) = V_D$;

Step 2: foreach($M_j \in H$) let $\lambda_j(n) = \frac{1}{E_{\beta_\mu^j}(\overline{T}_t^{M_j}(n))}$;
$C = \sum_{M_j \in H} \lfloor \mathit{\Delta}_n \cdot \lambda_j(n) \rfloor$;

Step 3: if($C < card(V)$)
{ for(*each* $M_j \in H$)
let $F_n^{-1}(M_j) \subset V(n)$ such that $card(F_n^{-1}(M_j)) = \lfloor \mathit{\Delta}_n \cdot \lambda_j(n) \rfloor$;
$V(n+1) = V(n) \setminus \bigcup_j F_n^{-1}(M_j)$;
$n = n+1$; goto 2;
}
else /* last step */
{ $\mathit{\Lambda} = \sum_{M_j} \lambda_j(n)$;
foreach($M_j \in H$) $\varrho_j = \frac{\lambda_j(n)}{\mathit{\Lambda}}$;
foreach($M_j \in H$)
let $W_j = F_n^{-1}(M_j)$ such that the set $\left\{ \left| \frac{card(F_n^{-1}(M_j))}{card(V(n))} - \varrho_j \right| \right\}$
is minimal over $\{W_j\}_{M_j \in H}$ and $\bigcup_j W_j = V(n)$;
}

Step 4: /*** start execution for the horizon Z ***/

The expected time of a single task execution on the machine M_k in each time step $n = \mu, \mu+1 \ldots$ is as follows:

$$E_{\beta_\mu^k}\left(\overline{T}_t^k(n)\right) = E_{\beta_\mu^k}\left(\varphi_2^k \circ \varphi_1^k(X_n^k)\right)) = \begin{cases} (1+\eta_k(j)) \cdot T_t^k & \text{dla} \quad n = \mu \\ \sum_{j=0}^K \left(\mathit{\Pi}_j^k(n) \cdot (1+\eta_k(j)) \cdot T_t^k\right) & \text{dla} \quad n > \mu \end{cases} \tag{20}$$

so the expected cost for one step $C_n(\beta_\mu, s_n, u)$ of the realized policy $u = \{F_n\}_{n=\mu,\mu+1,\ldots}$ is as follows:

$$C_n(\beta_\mu, s_n, u) = \qquad (21)$$
$$\begin{cases} \max_{F_n(M_k)} \left\{\nu^k(n) \cdot (1+\eta_k(j)) \cdot T_t^k\right\} & \text{for} \quad n = \mu \\ \max_{F_n(M_k)} \left\{\nu^k(n) \cdot \sum_{j=0}^{K^k} \left(\Pi_j^k(n) \cdot (1+\eta_k(j)) \cdot T_t^k\right)\right\} & \text{for} \quad n > \mu \end{cases}$$

where $s_n = (X_n^1, \ldots, X_n^m)$, $\nu^k(n) = card(F_n^{-1}(M_k))$ and since u is an *admissible policy*, $\forall k = 1, \ldots, m$ functions $F_n^{-1}(M_k)$ satisfy the constraint (9).

6.2 The Closed Loop Control

The first decision in the starting time epoch τ_μ in the *closed loop* algorithm is analoguous to *open loop* algorithm (the choice of action is based on the average states of the background load of each machine). The choice of the consecutive decisions (concerning the task subsets allocation) is based on the forcasted average states of the whole workload for each machine in H ($\overline{\Pi}^k(n)$ distributions) generated in each time epoch τ_n, $n = \mu+1, \ldots, \mu+Z-1$. In other words, in each time epoch τ_n, the choice of action is based on the generated forecasted computation time of elementary task only for the next step and successively repeated in each time epoch.

Algorithm 2

Step 1: $V(n) = V_D$;
foreach($M_j \in H$) let $\lambda_j(\mu) = \frac{1}{E_{\beta_\mu^j}(\overline{T}_t^{M_j}(\mu))}$;
foreach($M_j \in H$) /* computation is based on $\Pi^j(\mu)$ */
let $F_\mu^{-1}(M_j) \subset V(\mu)$ such that $card(F_\mu^{-1}(M_j)) = \lfloor \Delta_\mu \cdot \lambda_j(\mu) \rfloor$;
/*** start execution in the Δ_μ ***/
$n = \mu + 1$;
$V(n+1) = V(n) \setminus \bigcup_j F_n^{-1}(M_j)$;

Step 2: foreach($M_j \in H$) /* computation is based on $\overline{\Pi}^j(n)$ */
let $\lambda_j(n) = \frac{1}{E_{\beta_\mu^j}(\overline{T}_t^{M_j}(n))}$;
$C = \sum_{M_j \in H} \lfloor \Delta_n \cdot \lambda_j \rfloor$;

Step 3: if($C < card(V)$)
{ foreach($M_j \in H$)
let $F_n^{-1}(M_j) \subset V(n)$ such that $card(F_n^{-1}(M_j)) = \lfloor \Delta_n \cdot \lambda_j(n) \rfloor$;
/*** start execution in the Δ_n ***/
$V(n+1) = V(n) \setminus \bigcup_j F_n^{-1}(M_j)$;
$n = n + 1$; goto 2; }
else /* last step */
{ $\Lambda = \sum_{M_j} \lambda_j(n)$;

foreach($M_j \in H$) $\varrho_j = \frac{\lambda_j(n)}{\Lambda}$;
foreach($M_j \in H$)
let $W_j = F_n^{-1}(M_j)$ such that the set $\left\{ \left| \frac{card(F_n^{-1}(M_j))}{card(V(n))} - \varrho_j \right| \right\}$
is minimal over $\{W_j\}_{M_j \in H}$ and $\bigcup_j W_j = V(n)$;
}

The expected time of a single task execution on the machine M_k in each consecutive time step $n = \mu+1, \mu+2 \ldots$ is as follows:

$$E_{\beta_\mu^k}\left(\overline{T}_t^k(n)\right) = E_{\beta_\mu^k}\left(\varphi_2^k \circ \varphi_1^k(\overline{X}_n^k))\right) = \sum_{j=0}^{K}\left(\overline{\Pi}_j^k(n) \cdot (1+\eta_k(j)) \cdot T_t^k\right) \quad (22)$$

so the expected cost for one step $C_n(\beta_\mu, s_n, u)$ of the realized policy $u = \{a_n\}_{n=\mu+1,\ldots} = \{F_n\}_{n=\mu+1,\ldots}$ is as follows:

$$C_n(\beta_\mu, s_n, a_n) = \max_{a_n^k \in A_{s_n^k}^k}\left\{\nu(a_n^k) \cdot \sum_{j=0}^{K}\left(\overline{\Pi}_j^k(n) \cdot (1+\eta_k(j)) \cdot T_t^k\right)\right\} \quad (23)$$

where $s_n = (s_n^1, \ldots, s_n^m)$, $\nu^k(n) = card(F_n^{-1}(M_k))$, $a_n = (a_n^1, \ldots, a_n^m)$, and because u is an *admissible policy*, so $\forall n \ \ \forall k \ \ a_n^k$ has to satisfy constraint (9). For the first time step (i.e. for $n = \mu$) the expected time of a single task execution on machine M_k is defined by (20) and the expected cost for the first step of realized policy is defined by (21).

7 Conclusions and Future Works

Having based on the made assumptions regarding the model of heterogeneous computer network and distributed application (see Sect. 1), was defined stochastic model which describes the dynamics of the background load for the computers in the network (see Sect 3). Then, there were formulated two control problems for the task allocation, first one based on the presented stochastic model of background load and the second one based on the theory of Markov Decision Processes (see Sect. 4 and 5).

Next, there were defined two stochastic policies (algorithms) which are leading to the fastest execution of the distributed application. First algorithm was presented for the case of the single task execution time forecast for the whole horizon of the calculations i.e. the control is determined only once at the initial time epoch - *open-loop control* (see Sect. 6.1). The second one was presented for the case of the dynamic control i.e. the control is dynamically determined for each time step and the single task execution time forecast is based on the actual state of the whole workload of computer and on the possible decision - *closed loop control* (see Sect. 6.2).

The *closed loop control* usually is better than the *open-loop* one in case of computation on the horizon consisting of more than one time period because

it provides the better estimation of computational power of each machine (i.e. power coefficients for each machine and power coeffcient for the network - see Section 6.1) for each time epoch during computation. The *open-loop* algorithm prepares estomation of the computational power for each time period only at the starting time epoch, so it is only forecast of the computational power.

At the starting time epoch both policies give the same control but in the next time epochs the workload of machines may change contrary to the workload forecast made at the beginning by the *open-loop* algorithm. The *closed-loop* can improve the control because it dynamically checks the actual state of machine workload and because the control is determined not only on the actual state of background load but also on decision made in given state.

The presented models and policies of task distribution can be utilized rather to the coarse grained large applications consisting of the great number of identical processes for example the phase of *matrix-formulation* in CBS computation (topological decomposition for the CAE computation) [3] [4] as well as for other problems like on-line learning [5] (*closed-loop control*). Moreover, prediction of the task time execution sholuld be done by the identical pattern task for each machine in the heterogeneous network and on this basis should implement the allocation of the tasks of the application.

The worker tasks communicate with the manager task under the random network load, so communication workload of the network could be estimated in the stochastic way too. Threrefore, the analogous model of network load and algorithm of stochastic control for the process distribution in the heterogeneous network will be defined. Moreover, the pair of background load of the machine and the network load could be used to define the modification of the algorithm of the task distribution and to define the new cost function in each time epoch (*closed-loop control*).

References

1. Sinnen, O.I.: Task Scheduling for Parallel Systems. Wiley-Interscience, John Wiley & Sons, Inc. (2007)
2. Astrom, K.J.: Introduction to Stochastic Control Theory. Dover Publicztions (2006)
3. Onderka, Z., Schaefer, R.: Markov Chain Based Management of Large Scale Distributed Computations of Earthen Dam Leakages. In: Palma, J.M.L.M., Dongarra, J. (eds.) VECPAR 1996. LNCS, vol. 1215, pp. 49–64. Springer, Heidelberg (1997)
4. Onderka, Z.: Stochastic Control of the Distributed Scalable Applications. Application in the CAE Technology, Doctoral Thesis, AGH Univ. of Science and Technology, Department of Computer Science, Krakow, Poland (1997)
5. Even-Dat, E., Kakadey, S.M., Mansour, Y.: Online Markov Decision Processes. Journal Mathematics of Operations Research 34 (2009)
6. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics (2005)
7. Lepiarz, M., Onderka, Z.: Agent System for Load Monitoring of the Heterogeneous Computer Network. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 364–368. Springer, Heidelberg (2002)

Distributed Collaborative Visualization on Mobile Devices Using Interactive Video Streaming Techniques

Maciej Panka[1], Michal Chlebiej[2], Krzysztof Benedyczak[2,3], and Piotr Bała[2,3]

[1] UCNTN Nicolaus Copernicus University, Torun, Poland
maciej.panka@umk.pl
[2] Faculty of Mathematics and Computer Science Nicolaus Copernicus University, Torun, Poland
[3] ICM, University of Warsaw, Warsaw, Poland

Abstract. Remote visualization using mobile devices has been a challenge for distributed systems for a long time. Large datasets, usually distributed on different servers require high network bandwidth and significant computational power for effective, real time rendering. The problem is getting more complex when data are visualized in collaborative environment, where every user can interactively participate in rendering session.

In this paper we present a distributed system we have developed for the interactive visualization of remote datasets on variety of mobile devices such as laptops, tablets and cell phones. In our system mobile users can join sessions, where they can collaborate over remote data in real time. Every user can watch presentation or can become presenter. If needed, users can individually manipulate the data without affecting rest of participants.

During these sessions all the data are generated on dedicated rendering servers, compressed on-the-fly by the encoding machines using video codec and progressively sent to participants as video streams. Every video stream is dynamically adapted to individual capabilities of users' devices and their network bandwidth. Our system works in a distributed environment, where every machine serve different functionality, like data storage, frames rendering or video compression. Successive parts of processed data are streamed between different servers in real time to achieve highly interactive visualization with minor latency. Based on this model we took off most of the computational power from client's application so it can be run on almost any kind of modern mobile device. We were also able to achieve very high video quality and frame rates. System can work with 2D, 3D and even animated 3D data, all of them being processed remotely in real time. At the end of this paper we present some preliminary results of performance test we have obtained using sample multidimensional datasets.

Keywords: distributed data visualization, mobile visualization, mobile collaboration, distributed video streaming.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 191–200, 2012.

1 Introduction

Pervasive computer technologies have redefined group collaboration long time ago. Modern collaboration techniques rely mostly on the Internet and mobile communication systems, which provide almost unlimited access to the global information resources. Presently available solutions for computer-based cooperation could be divided into four categories depending on a place-time relation of participating users, as presented in [1] (same time - same place, same time different place, different time same place, different time different place). The collaboration, which involves simultaneous presence of all users is usually called synchronous communication. Otherwise it is called asynchronous communication. Most popular examples of today's asynchronous systems are email and discussion boards. On the other hand, exemplary synchronous techniques would be instant messaging, chat, shared boards, teleconferencing or videoconferencing.

Modern collaboration techniques could also be effectively adopted into the scientific activities. Progressive globalization causes, that many scientific projects involve cooperation of researchers from different departments, universities or even countries. With the use of the Internet and computer technologies the collaboration is much easier today then it used to be in the past. However, in some situations it is still a challenge to virtually bridge the distance between participants, especially when the collaboration is centered on the large datasets, distributed among storage centers around the globe. Many modern scientific experiments and simulations are so complex, that they must be realize on dedicated computer clusters. Obtained data volumes are usually so large that they cannot be easily transferred between distant users, and must be stored on the dedicated servers.

One of the biggest challenges in this area is an effective visualization of these data through the Internet, without the need of downloading and rendering it on local computers. The problem is even bigger when the visualization should be realized on mobile devices, which still don't have enough computational power to effectively render complex data in real time. In this paper we present a framework, which confronts this problem allowing mobile visualization of remote datasets in a distributed collaborative environment. Our framework consists of a distributed server side application, as well as the client's application, which could be run on thin, mobile devices. According to the users' demands, server application reads the data from adequate storage area, renders it in real time, compresses using a video codec and broadcast to client using a dedicated streaming protocol. Remote users receive these streams, decode them and display on the screen. Moreover, distant users can collaborate over the remote data in real time, with the use of a built-in teleconferencing functionality and a shared, interactive video area. If needed, they can also manipulate remote data without affecting rest of the session participants.

The rest of this paper is organized as follows. Section 2 briefly describes related works covering mobile visualization in a distributed collaborative environment.

In the sections 3 and 4 we describe in details main functionalities of the system, its architecture and technologies that we have used to implement it. Section 5 presents system's performance test results obtained during a sample collaborative session. Section 6 concludes this paper and draws up further work.

2 Related Work

There are few different approaches to the remote visualization problem. First category involves systems, which force data rendering on clients' devices. Many solutions which are based on this model compress all the data on the server and transfer it progressively to distant users using different kind of 3D objects streaming techniques, like progressive meshes or levels of details [2, 3, 25–27]. Other commonly used approaches are based on the Virtual Reality Modeling Language, which is a text file standard for 3D scenes representation [4, 5]. Although, one of the biggest challenges when using VRML standard is the size of generated files, which causes some serious limitations in a data network transfer. References [6] and [7] propose different compression techniques, which reduce this latency.

Second category involves systems, where all the data is rendered on dedicated server and transferred to client as a digital image sequence. References [8] and [9] introduce exemplary solutions based on the image streaming techniques, which make use of the lossless compression algorithms, like LZO or BZIP. The authors of [10] and [11] presented possibilities of using JPEG 2000 compression, which increases the overall image compression level. They also showed, that with the use of the Motion JPEG 2000 it is possible to visualize animated 3D data, although there are more sophisticated techniques available in this area, like for example VNC [12] or by means of a dedicated video codec [13, 14].

The idea of collaborative visualization of remote data has been described in details in [1]. Reference [15] proposes exemplary solution to this problem by means of establishment of dedicated rooms, equipped with the specialized videoconferencing hardware and software. However, a much ubiquitous solution would be to use participants' individual computers during the collaborative sessions, including modern mobile devices. Exemplary systems based on the VRML text files were introduced in [6], [16], [17] and [18]. The authors of [19] and [20] introduced sample frameworks for a collaborative visualization, which render all the data on a server and send it to every connected user as a sequence of digital images. The references [21], [22] and [23] additionally made use of dedicated video codecs.

In this paper we propose a distributed visualization system, which derives directly from the video streaming techniques, allowing thereby an effective and fully interactive data visualization on different kind of mobile devices, including tablets and cell phones.

3 System Overview

3.1 Remote Collaboration

The system consists of the client and server applications. Server modules are responsible for session management, clients' collaboration, data rendering and video encoding. On the other hand, client's application's main job is to receive a video stream, decode it and display on the screen. During collaborative sessions clients can additionally communicate using a built-in teleconferencing module.

The first user, which connects to the server, becomes a session presenter. Successive users can either start their own sessions or they can join previously established one. The presenter of each session can load the remote data and manipulate it in real time using his mouse or touch gestures. As the response to user's requests server's application loads the data from the source, renders it, compresses using a video codec and broadcast to every user participating in the session. By default all the other session participants can only passively display the video stream on their screens, listening to the presenter's lecture at the same time. At any point during the session, the presenter can pick up any connected participant, giving him temporary ability to manipulate the data and to speak to other users. From this moment both the presenter and selected user can discuss among other participants using teleconferencing module. Additionally, during the whole session users can individually manipulate the remote data, having a chance to analyze it from different angles, without affecting other participants.

3.2 Interactive Visualization

Our system can work with a 2D, 3D and animated 3D data, all of them being processed on the server and broadcasted to the users in real time. When the presenter changes his view angle, the remote data is dynamically processed by the adequate servers, which generate successive frames and compress them using a video codec. Video frames are streamed to all participants in real time, giving them effect of a zoom, move and spin animations [Fig. 1].

The output of a two-dimensional data is a typical digital image. Depending on the data source, these images could have very large resolutions, transcending typical screen sizes of mobile devices. Our system dynamically crops and resizes every image, fitting it to the individual capabilities of users' devices. The session presenter can zoom in / out and move to the different parts of an image by selecting its successive regions on the screen of his device. According to the user's actions these events are transferred to the server application, which crops adequate parts of an input image and places them on an encoding sequence. Successive frames are compressed and streamed to the users one by one, giving the effect of an image motion. Every session participant receives his own video stream, individually customized to the capabilities of his device, including a screen size and network parameters. That means, that depends on the number of concurrent connections every server could encode in parallel many different video streams during a single visualization session. The visualization of a 3D data additionally involves rotation

of a remote object over the X and Y axes. As a response to the presenter's rotation events, adequate server application generates successive frames in real time, which are later encoded and progressively broadcasted to users. During the visualization of an animated 3D data all the above techniques work in a loop, giving thereby an effect of data evolution in time.

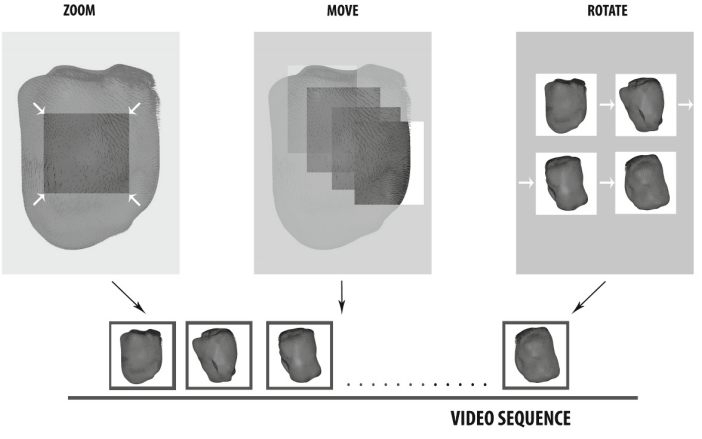


Fig. 1. Successive frames of processed data are encoded as a sequence, and broadcasted to all participants as an independent video streams, individually customized to their devices

4 System Architecture

The server side application consists of three modules: a session manager, data renderer and video encoder. To achieve the highest possible performance level, each of them should be run on a different machine, so they could process their tasks in parallel. For a better load balancing during multiuser sessions, the rendering and encoding modules, which are the most CPU consuming parts of the system, should be additionally distributed between servers [Fig. 2].

Every joining user starts his session by connecting to the manager module. The session manager stores the information about other servers available in the system, together with their current load, including avarage CPU / RAM usages and a number of concurrent users. Every server measures these parameters and sends them to the session manager periodically. Based on this information the session manager is able to find the less laden servers at the moment (encoding and rendering modules), and sends their IP addresses to the newly connected presenter. With the use of these parameters the presenter is able establish the connection with the selected encoder, which from this moment controls the rest of the visualization computes. The further participants who join this session establish their connections only with the session manager module, which also acts as a video proxy between the encoding module and clients' applications.

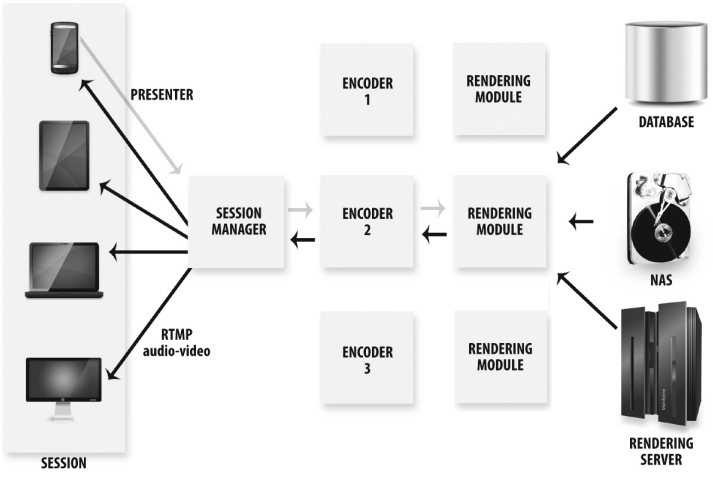


Fig. 2. A general schema of the system's communication workflow. To achieve highly interactive visualization, the session manager, encoder and rendering modules exchange partial data progressively, and transfer them to distant users as a live video stream.

By default, every new user is able only to watch the presentation passively, so there is no need for them to communicate with any of the encoders. The only situation when other users connect to the encoder is when they become temporary presenters or when they want to individually manipulate the remote data. The selection of the less laden server in that case is realized the same way as described earlier.

In a response to the presenter's activities (zooming, moving and rotating) client's application sends adequate events to the selected session encoder. Based on these events the encoder creates appropriate frame sequence and progressively sends it to session participants. During the two-dimensional data visualization, all the images are buffered directly on the encoder machine, as there is no need for permanent data rendering. However, working with the 3D and animated 3D data involves parallel image generation in the rendering module, which broadcasts them to the encoder one by one in real time (rotation and animation frames). The rendering module is completely transparent to the data source, and it could be easily adapted to cooperate with a database, storage drive or a dedicated rendering machine.

Regardless from the dimension of the data source, the encoder processes every frame on-the-fly, based on the current parameters of the presenter's view (crops and resizes), compresses them using a video codec and broadcast to the session manager, which republishes them to the rest of the session participants. To serve many users simultaneously the encoding module runs a multithread application, which in parallel compresses and broadcasts different video streams to session participants (different resolutions, bit rates and qualities of the outgoing video). We are not sending the videos directly to the session users, because practically

in most cases one stream would have many simultaneous recipients, which have devices with the same screen sizes and similar network bandwidth. This approach allowed us to limit number of the encoding threads being run in parallel, reducing thereby the encoding server's load.

Session users can communicate using a teleconference module, which has been built into the manager server's application. The audio data, which is transmitted from the presenter's device, behaves exactly the same way as other videos streamed to the session manager from the encoder module. Client's application acquires the data from user's microphone, compresses it and broadcasts to the session manager, which re-streams it to the rest of the session participants.

All the server side modules have been written in Java. The session manager's application has been deployed on the Wowza Media Server, which is currently one of the most powerful video streaming solution. The WMS assures communication with the sessions' participants and provides the video re-streaming solution to a variety of popular protocols, including RTMP, RTSP or HTTP.

Rendering module streams successive frames to the encoder using a TCP socket connection. During a typical visualization session both machines exchange large amount of data, so it is recommended that they communicate using a broadband connection. We have decided not to use any compression techniques at this point because they would require additional computational power from the rendering module. Encoding module compresses the video sequence and streams it to the session manager with the use of FFMPEG, which is currently one of the best open source solutions to handle digital video. Encoded video sequence is streamed to the session manager, and later to all users by means of the Adobe's Real Time Messaging Protocol.

5 Results

We have run a series of performance tests of the system. At first we have measured the video compression speed and CPU usage of the encoder. The session management module was run on an Intel Xeon 5050 3GHz, the rendering module on a dual core Intel Xeon X5355 2.6 GHz, and the encoding module on a quad core Intel Xeon E5420 2.5 GHz. All the servers were connected using 1 Gb network.

We have run four tests using different types and number of clients' devices, each of them having set different screen resolutions. That way we were able to measure the system's performance during simultaneous encoding of multiple video streams. At first we have tested its behavior using three different mobile devices: a laptop running Mac OS X, tablet and a cell phone, both equipped with the Android 2.2 system. Three other tests were run on a Windows Vista desktop computers, which involved 5 and 10 simultaneous users, each of them having different screen sizes. Additionally, 5 users test was run two times. The first test was run for a group of devices equipped with the small screens, and the second effort, which involved only a high resolution video encoding. Table 1 presents obtained data.

Table 1. Encoding speed averages measured in frames per seconds during four different collaborative sessions

Resolution	3 users	5 small screens	5 big screens	10 users
320 x 240	-	228	-	119
400 x 240	-	228	-	116
640 x 480	-	86	-	40
800 x 480	78	84	-	40
854 x 480	-	52	-	37
768 x 576	-	-	63	31
800 x 600	-	-	37	30
1024 x 600	57	-	36	26
1024 x 768	41	-	27	20
1366 x 768	-	-	26	19

The results show, that a single encoding server is able to effectively compress many simultaneous video streams, all set at different resolutions. From the users point of view the lowest acceptable encoding speed is 15 frames per second. Below that value the visualization looses its smooth and decreases the overall reception of the session. In our experiment all results were above the minimum fps, regardless from the targeted users device (desktop, tablet, cell phone). The CPU usages of the encoding server were 25%, 30%, 50% and 75% respectively, never reaching the maximum capability of the system.

During the experiment we have also measured the network communication latency between the clients and a session manager modules. We have also tested the overall delay between client's activity start (mouse or touch gesture), and the moment when the first visualized video frame was received from the server (including image generation in the rendering module, compression in the encoder and all the network transfers). Both tests were run for 5 and 10 simultaneous users connected to a single session, using single rendering and encoding machines. Every group was tested three times, using different screen resolutions of the clients' devices: 320x240, 800x480 and 1366x768.

The average measured RTMP communication latency was 0.05 ms and with this level it didn't affect the overall system efficiency. The overall encoding time of a single video frame was also gratifying. For the typical mobile devices' resolutions (320x240 and 800x480) the visualization's latency was lower than 100 ms in most cases. Only when 1366x768 resolution was set and 10 simultaneous users were connected to a single session, the encoding latency increased to 300 ms. However, even then it did not affect the overall comfort of the visualization session.

6 Conclusion and Further Work

In this paper we presented the system we have developed for a distributed collaborative visualization of remote datasets. The system allows for visualization of a multidimensional data on a variety of modern mobile devices. Distance users can join sessions and collaborate over the remote data in real time using a built-in teleconferencing system. They can also manipulate the data individually, without affecting rest of the participants. Obtained results of the performance tests showed, that thanks to the distributed architecture our system is able to effectively serve many simultaneous participants, even if they are using different devices.

In the future we are planning to experiment with the multicast and peer-to-peer communications, which we believe, in some cases should further increase the overall efficiency of our system. Beside the many laboratory tests that have been taken so far, one of the most important thing is also the evaluation of the users' perception of the system. The current results of the latency in communication between the servers and mobile users show, that the fluency and interactivity of the remote visualization could be nearly the same, as if the data were stored locally. We are planning to do much bigger researches in this area soon, once the final version of the application's user interface will be prepared.

References

1. Brodlie, K.W., Duce, D.A., Gallop, J.R., Walton, J.P.R.B., Wood, J.D.: Distributed and Collaborative Visualization, pp. 1–29. The Eurographics Association and Blackwell Publishin (2004)
2. Hu, S.: A Case for 3D Streaming on Peer-to-Peer Networks. In: Web3D 2006. The Association for Computing Machinery, Inc. (2006)
3. Sung, W., Hu, S., Jiang, J.: Selection Strategies for Peer-to-Peer 3D Streaming. In: NOSSDAV 2008 (2008)
4. Mosmondor, M., Komericki, H., Pandzic, S.: 3D Visualization of Data on Mobile Devices. In: IEEE MELECON 2004 (2004)
5. Lipman, R.R.: Mobile 3D visualization for steel structures. Automation in Construction 13, 119–125 (2004)
6. Engel, K., Ertl, T.: Texture-based Volume Visualization for Multiple Users on the World Wide Web, http://www.vis.uni-stuttgart.de/ger/research/pub/pub1999/EGVE99.pdf
7. Zhou, H., Qu, H., Wu, Y., Chan, M.: Volume Visualization on Mobile Devices, http://www.cse.ust.hk/~huamin/pg06_mobilevis.pdf
8. Constantinescu, Z., Vladoiu, M.: Adaptive Compression for Remote Visualization. Buletinul Universitatii Petrol-Gaze din Ploiesti LXI(2), 49–58 (2009)
9. Ma, K., Camp, D.: High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network. IEEE (2000), http://www.cs.ucdavis.edu/~ma/papers/sc2000.pdf
10. Dragan, D., Ivetic, D.: Architectures of DICOM based PACS for JPEG2000 Medical Image Streaming. ComSIS 6(1) (June 2009)
11. Lin, N., Huang, T., Chen, B.: 3D Model Streaming Based on JPEG2000, http://graphics.im.ntu.edu.tw/docs/tce07.pdf

12. Stegmaier, S., Magallon, M., Ertl, T.: A Generic Solution for Hardware-Accelerated Remote Visualization. In: IEEE TCVG Symposium on Visualization (2002)
13. Cheng, L., Bhushan, A., Pajarola, R., Zarki, M.: Real-Time 3D Graphics Streaming using MPEG-4, http://vmml.ifi.uzh.ch/files/pdf/publications/3DMPEG4.pdf
14. Noimark, Y., Cohen-Or, D.: Streaming Scenes to MPEG-4 Video-Enabled Devices. IEEE Computer Graphics and Applications (January/February 2003)
15. Childers, L., Disz, T., Olson, R., Papka, M.E., Stevens, R., Udeshi, T.: Access Grid: Immersive Group-to-Group Collaborative Visualization, http://www.ipd.anl.gov/anlpubs/2000/07/36282.pdf
16. Knodel, S., Hachet, M., Guitton, P.: Visualization and Interaction with Mobile Technology. In: MobileHCI 2008 (2008)
17. Wang, M., Fox, G., Pierce, M.: Grid-based Collaboration in Interactive Data Language Applications, http://grids.ucs.indiana.edu/ptliupages/publications/GridCollabIDL_ITCC2005.pdf
18. Manssour, I.H., Freitas, C.M.D.S.: Collaborative Visualization in Medicine. In: WSCG 2000 (2000)
19. Engel, K., Sommer, O., Ertl, T.: A Framework for Interactive Hardware Accelerated Remote 3D-Visualization, http://www2.ccc.uni-erlangen.de/projects/ChemVis/VisSym2000.pdf
20. Lee, S., Ko, S., Fox, G.: Adapting Content for Mobile Devices in Heterogeneous Collaboration Environments, http://grids.ucs.indiana.edu/ptliupages/publications/icwn03.pdf
21. Goetz, F., Domik, G.: Remote and Collaborative Visualization with openVISAAR, http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Domik/publications/Remote_and_Collaborative_Visualization_with_openVisaar_VIIP_2003_.pdf
22. Engel, K., Sommer, O., Ernst, C., Ertl, T.: Remote 3D Visualization using Image-Streaming Techniques, http://www.vis.uni-stuttgart.de/ger/research/pub/pub1999/ISIMADE99.pdf
23. Hereld, M., Olson, E., Papka, M.E., Uram, T.D.: Streaming visualization for collaborative environments, http://www.mcs.anl.gov/uploads/cels/papers/P1512.pdf
24. Adobe Inc., Open Screen Project, http://www.openscreenproject.org/
25. Pajarola, R., Rossignac, J.: Compresed Progressive Meshes. IEEE Trans. Vis. Comput. Graph. 6(1), 79–93 (2000)
26. Chen, Z., Bodenheimer, B., Barnes, J.F.: Robust Transmission of 3D Geometry over Lossy Networks. In: Conf. on 3D Web Technology (2003)
27. Kim, J., Lee, S., Kobbelt, L.: View-dependent Streaming of Progressive Meshes (2004)

P2P Approach to Knowledge-Based Dynamic Virtual Organizations Inception and Management

Marcin Stelmach[1], Bartosz Kryza[2], and Jacek Kitowski[1,2]

[1] AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Electronics,
Department of Computer Science, Al. A. Mickiewicza 30, 30-059 Krakow, Poland
[2] Academic Computer Centre CYFRONET AGH
ul. Nawojki 11, 30-950 Krakow, Poland
{stelmach,bkryza,kito}@agh.edu.pl

Abstract. For several years, all major communicational and informational aspects of organizations are being virtualized. This trend can be expected to evolve even further, provided sufficient technology is available, which can foster this process. Several research efforts have dealt in the past with the problem of creating and supporting management in Virtual Organizations, i.e., introducing the possibility of virtualizing the entire organization concept, by means of special IT infrastructure. One of the main problems of this approach is allowing the organizations to discover and collaborate with other organizations. In this paper we analyze the possibility of using P2P technology to data knowledge distribution in distributed systems. We present a survey and comparison of various structured and unstructured P2P networks. We describe our P2P approach to knowledge based emergent Virtual Organizations inception and management.

Keywords: P2P, Virtual Organization, ontology.

1 Introduction

The need for creating large scale distributed systems increased significantly in the last decade. This includes in particular need for significant computing power, and automation of processes such as collecting, processing, and storage of large amounts of data. Despite the advantages, they have to deal with the problem of resource and information discovery. One of the technologies designed to remedy this problem are the decentralized P2P systems, for instance Large Scale Disaster Information System based on P2P Overlay Network [26]. The distributed semantic approach is one of the elements of a future interconnection environment which will assist people in solving complex problems. In order to make it more versatile, support from knowledge management technologies is necessary. In our previous research we have created a Virtual Organization inception and management support framework called FiVO (Framework for Intelligent Organizations)

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 201–210, 2012.

[9], providing among others distributed contract negotiation, monitoring of VO execution and security enforcement. The backbone of the system is a distributed knowledge base, called Grid Organizational Memory (GOM) [10], designed to support dynamic Virtual Organization (VO) inception and management using Web Ontology Language (OWL) [1] ontologies. They include semantic descriptions of service classes, functionality and performance records. More specifically, GOM contains overall semantic descriptions of services, resources, applications and data. The most important element in the application of such systems as the GOM for VO management is easy and efficient searching for relevant information. An important factor is also scalability - easy addition of new resources and search speed. To ensure all these properties, we performed research on the application of P2P networks. With the amount of resources stored in the GOM up to several hundred thousand individuals we should decentralize it to avoid bottlenecks. Since the initiation of research on a possible merger of the Semantic Web and P2P networks has given rise to several projects such as Ontology-based Information Service (DIS) [22] or the OpenKnowledge framework [5]. The process of sharing resources and services defines a model for building large scale computing or information systems. In this paper we present P2P approach to knowledge based emergent Virtual Organizations inception and management. The paper is organized as follows. The work related with this area of research is described in section 2. In Section 3 P2P technologies description in the area of structured and unstructured P2P networks is given. Section 4 presents an application of the Grid Organizational Memory. In section 5 we describe our P2P approach to knowledge based emergent Virtual Organizations. Section 6 presents results from scalability and reliability tests of the system. Finally, Section 7 outlines summary with some conclusions.

2 Related Work

One of the first projects devoted to the problem of ontologies distribution in P2P networks was DRAGO [19]. It enables distribution, creation, modification and deletion of ontologies and making queries on these ontologies. In this system every peer called DRP (DRAGO Reasoning Peer) owns in its local memory the whole information about the available ontologies (ontology ID and its physical address). As a contrary, in the PeerThing [3] architecture, peers are split in groups which are merged later in one large network. Each pool owns one "Poolhead" responsible for connecting with other groups in P2P network. After receiving the query, Poolhead decides if local clients know the answer or it whether should be sent to other groups. Edutella [15] project is the precursor of unstructured P2P network usage in the semantic technologies. It enables various systems to form networks for metadata exchanging according to semantic web standards. Unfortunately, information searching was not as efficient as for the systems based on structured architecture with DHT. In the paper [8] a detailed data management strategy for a DHT based RDF store providing reasoning, robustness and load-balancing is presented. In [25] the solution for distributed

reasoning using the technique for materializing the closure of an RDF graph based on MapReduce [2] is presented. The optimizations for encoding the RDFS ruleset in MapReduce were found. Finally, some systems were designed such as Bibster [7] focusing in the knowledge representation of the digital content via semantic based techniques in P2P networks.

Our work explores how P2P technologies can be used to distribute the information in VO using GOM knowledge base. The suggested architecture, as contrasted to the presented solutions, is based on clustering of resources in an appropriate way and efficient querying. Our approach has an excellent scalability - adding of new resources to the network is straightforward. Peers do not implement complex algorithms as in the structured networks. Our approach is dedicated for VO contexts, where participating organizations publish information about their resources and activities, and it will not be optimal for projects with different information characteristics.

3 P2P Technologies

P2P systems are based on general principle that each node linked to the network, behaves as a client and a server. The P2P overlay network consists of all the peers participating as network nodes. Based on how the nodes in the overlay network are connected to each other, we can classify the P2P networks as structured or unstructured.

Structured P2P networks employ a globally consistent protocol to ensure that nodes can route a search to some peer that owns the desired file. Such a guarantee necessitates a more structured pattern of overlay connections. All structured P2P systems modelled as Distributed Hash Tables (DHT) assign identifiers to nodes, typically at random. Random selection in DHTs can be done by randomly choosing a value from the DHT number space, and routing to this value. The nodes of the DHT ring maintain connections in the DHT ring in terms of specific geometry mode, e.g. Bamboo DHT [16] or Pastry DHT [18]. An unstructured P2P network is created when the overlay connections are established arbitrarily. If a peer wants to find a desired piece of data in the network, the query must be flooded through the network to find as many peers sharing the data as possible. These systems use the flooding technique to broadcast the requests of the resources in the network. To limit the vast number of messages produced by flooding, many techniques, such as TTL (Time-To-Live) tags, dynamic querying, random walks and multiple random walks [24], were proposed. Several popular unstructured P2P applications, such as FastTrack [12] and Gnutella [17] explored organizing of the resource nodes in the superpeer mode to improve the performance of the search. These systems forward queries to high-capacity superpeers chosen due to their capability to process large numbers of requests. A superpeer behaves as a local search engine, building an index of the files being shared by each connected peer and proxying search requests on behalf of these peers by flooding requests to other superpeers. Our structure, which will be based on this

model, is an unstructured P2P networks, which are easier to create and use. Such networks can be quickly constructed, because peers do not need to implement advanced DHT algorithms.

4 Grid Organizational Memory

Grid Organizational Memory (GOM) [10,11] is a versatile distributed knowledge base for managing semantic metadata in distributed computing environments such as Grid or SOA based infrastructures. GOM allows to separate the metadata space into parts which can be managed by separate GOM Engine processes deployed on different machines in OWL format. Each GOM deployment can have a GOM registry process which keeps track of currently running GOM instances and can distribute queries and events to the knowledge base. The knowledge base allows for efficient storage and querying for the metadata, but also provides a semantic protocol called OWL Change for fine grained modifications of the underlying ontologies. This allows the knowledge base to record the changes of the ontologies, which represent the evolution of knowledge. One of the main applications of GOM is to support dynamic VO inception and management, within a larger software framework called FiVO [9]. In FiVO different aspects of knowledge related to both the physical organizations (e.g. resources, employees, capabilities) as well as aspects related strictly to VO (e.g VO contract, VO monitoring information) can be divided into logical parts and distributed over the IT infrastructure of the organizations.

5 Peer-to-Peer Overlay for GOM

After analyzing the existing P2P solutions, we chose an unstructured P2P network solution based on the use of super nodes (JXTA) [6]. Due to the topology of rendezvous peers connections (ring type) and the topology of regular peers (star type) which refers to VO structure, we chose JXTA. The JXTA protocols are defined as a set of XML messages which allow any device connected to a network to exchange messages and collaborate independently of the underlying network topology. JXTA architecture has several aspects which distinguish it from other models of distributed networks: resources are described using XML abstraction of connections between nodes independent of the central addressing system and distributed search infrastructure based on DHT (Distributed Hash Table) to the indexed stocks.

5.1 P2P Network Architecture

We based our architecture on the mechanism of Shared Resource Distributed Index (SRDI) [23], which is used by JXTA. One can easily promote ones query in the network. Each rendezvous node (the node is used to promote and search for resources and maintains a list of nodes connected to it) builds indexes announcements published by the nodes. A node sends an index created by the

SRDI mechanism of ones rendezvous type node, which reduces significantly the number of nodes used to search for resources. In addition, when the index is sent to the rendezvous node type, node sends a copy of this index to other nodes such as rendezvous. Distributed replicas of the index ensure that information about resources is not lost when rendezvous node is removed. Each rendezvous node has its own list of other well-known peer-types of the same type, which is updated from time to time.

5.2 Peer Architecture

When designing the main network our assumptions were: universality (independence on used ontologies), support for conflicting resources, interconnections of resources from different nodes, and the query performance. In Figure 1 the architecture of a single node connected to a network is presented.

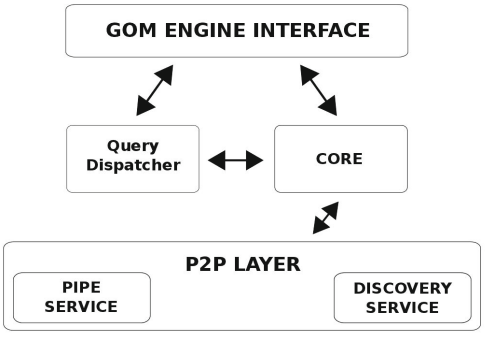


Fig. 1. Peer architecture

GOM Engine Interface - set of interfaces exposed to the GOM engine. They allow GOM engine to communicate with the P2P module.

Query Dispatcher - module responsible for analyzing queries, dividing them, etc. This module implements the algorithm in Listing 1.

Core - module responsible for communication between all layers and for providing information about classes held in the GOM ontologies

P2P Communication - is the communication layer, responsible for communication between nodes on a P2P network. In this layer, we can distinguish two components:

- **Pipe Service** - module is responsible for querying nodes for resources, sending the prepared questions and receiving answers from other nodes,
- **Discovery Service** - module is used to broadcast information about which resources are stored in GOM. When connecting to a network node, this module sends information about its resources to the node type Rendezvous. Module from time to time update this list.

5.3 Distribution of Resources

Distribution of resources is presented in Figure 2. The system supports the so-called collision resources (individuals with the same id must be unique but not necessarily contain the same data). Periodically, each node refreshes (rendezvous node sends to peer node) notices about available resources. Each node before connecting to the network makes a list of class names with individuals which are stored by its underlying GOM instance. Then, when a connection is made it announces the names through the Service Discovery module. Doing so allows to store current data on the resources of the network. Each node can to search

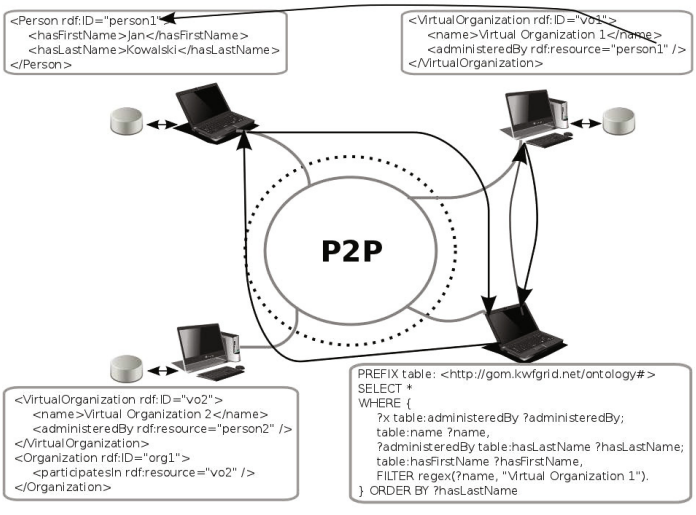


Fig. 2. Distribution of resources

the network. This is done using SPARQL language. However, each query before executing the algorithm in Listing 1 has to be decomposed using Algorithm 1. This algorithm is responsible for sharing queries, grouping and joining them. The `getClassesFromQuery()` method is responsible for delivering individuals types or domains and ranges of the properties used in the query. The other method called `getQueriesForClass()` is responsible for mapping of split queries to the appropriate classes found by `getClassesFromQuery()` method. The usage of the algorithm is necessary because it may happen that an individual class, connected through some relationship, is not in the same node as their relationship. The algorithm accelerates the effectiveness of the program by splitting queries and grouping them due to the classes of the searched individuals. The queries are sent in packets thus accelerating the search time. After receiving the results, they are combined and presented in the form of RDF. Sample query has been presented in the listing below:

Listing 1. Example query

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vbe: <http://fivo.cyf-kr.edu.pl/ontologies/VBE#>
PREFIX vo: <http://fivo.cyf-kr.edu.pl/ontologies/VOOntology#>
SELECT ?OrganizationName, ?OrganizationAddress, ?VOName
WHERE {
   {
      ?x vbe:administeredBy ?administeredBy; vo:VO.name ?VOName .
      ?administeredBy vo:Organization.name ?OrganizationName;
                      vo:Organization.address ?OrganizationAddress .
      FILTER regex(?VOName, "Virtual Organization 1") .
   } UNION {
      ?x vbe:administeredBy ?administeredBy ; vo:VO.name ?VOName .
      ?administeredBy vo:Organization.address ?OrganizationAddress .
      ?x vo:VO.name "Virtual Organization 2" .
   } UNION {
      ?x vo:Organization.name ?OrganizationName .
      ?x vo:Organization.address ?OrganizationAddress .
      ?x vo:Organization.name "Organization 1" .
   }
} ORDER BY ?OrganizationName
```

Algorithm 1. Grouping and decomposition algorithm

```
1: Input: query
2: Output: querymap
3: classes = getClassesFromQuery(query);
4: if query has objectProperty OR UNION then
5:       queries = decomposition(query);
6:       for i=0 to classes.length do
7:             querymap.put(class, getQueriesByClass(queries));
8:       end for
9: else
10:      querymap.put(classes[0], queries);
11: end if
12: sendQueries(querymap);
```

6 Performance Tests

To perform the required tests, 10 machines with Pentium M 1.86 GHz and 1,5 GB RAM were used. Each machine contained more then one program instance. One instance is equal to one node. The influence of the network was negligible in the carried tests. The correctness of the reasoning was verified by comparing the answers with the expected results. All tests were carried out by the following formula:

$$T = \frac{\sum_{i=1}^{N}(t_s - t_e)}{N} \qquad \begin{array}{l} t_s \text{ - the time of receipt of response,} \\ t_e \text{ - the time to send responses,} \\ \text{N - number of repetitions.} \end{array} \tag{1}$$

Questions were requested at the same time by 10 independent nodes. In Figure 3 a speed of the response time due to increasing resources and the nodes

in the network is presented. From the right plot in Figure 3 we can conclude that the optimal distribution of resources is always limited, the time of routing the messages is smaller than the analogous time for a single node. On the left plot in Figure 3 we presented the results of the response time for an increasing number of nodes at a fixed amount of resources in those nodes. Distribution of resources was also a more structured, e.g. appropriate nodes have much greater amount of resources "Person class" from the other nodes. The arrangement resembles the resources in the VO where organization provides to the other nodes its unique knowledge. In this approach, there is always the optimum, e.g. for 2 million individuals the best dispersion is obtained for 200 nodes. It should be noted that this is dependent on the choice of system hardware. In this case the Jena Framework [13] and Pellet [20] reasoner were applied. Figure 4 presents the search time on a single node. For a large number of individuals >200k query time was very large. As one can see, the time on a single node for a small number of individuals of the order of 10k is satisfactory, but for a greater number the time is too big. These results give an important conclusion - the use of P2P network performs best for large amounts of resources.

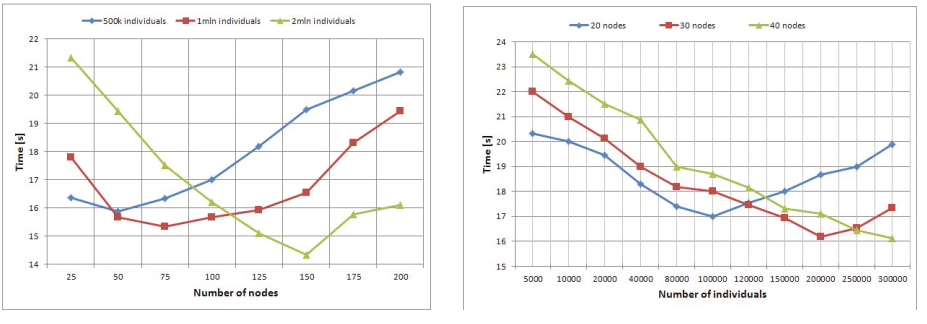


Fig. 3. P2P network time as a function of number of peers at the given number of individuals (left) and number of individuals at the given number of peers (right)

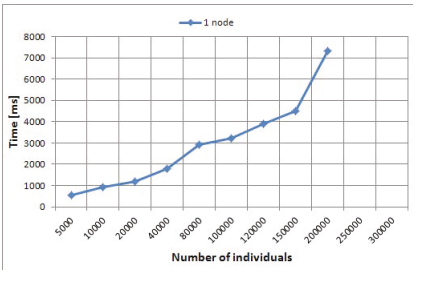


Fig. 4. Query time for a single peer

7 Conclusions and Future Work

In this paper we presented our approach to P2P distribution of ontological knowledge base. We showed P2P network diagram that was used. It is a network of distributed computers, highlighting is a compromise between the network with a central server and the distributed network. Our choice guarantees an access to every node and enables easier monitoring and managing of the network. Searching for resources is effective and does not clog the network links as distributed. Tests show that our approach is adequate for the envisioned application in VO management due to the inherent knowledge distribution in VO. Each organization has unique knowledge to share with other organizations. In our opinion, the solution will be applied in systems such as Grid due to a good VO support. Our approach can be applied to distribute VO resources in heterogeneous systems (monitoring systems [4], negotiation systems [21] or other similar systems [14]). We will also consider applying this approach to Cloud interoperability scenarios, where we could run or migrate user application between diffrent Clouds. It ensures the availability and consistency of data and can be applied to distributing the knowledge in order to select the Cloud that fits our application best. Test results showed that the data is distributed in a structured way. In terms of the data structure, the set of Clouds behaved similarly to VO. The security of P2P network from the standpoint of the VO is being currently examined in separate studies.

Acknowledgment. This research is partially supported by European Regional Development Fund program PL-Grid POIG.02.03.00-00-007/08-00.

References

1. OWL 2 Web Ontology Language Document Overview. W3C Recommendation (October 2009), http://www.w3.org/TR/owl2-overview/
2. Dean, J., Ghemawat, S.: Mapreduce: Simplified Data Processing on Large Clusters. Operating System Design and Implementation, 13 (2004)
3. Heine, F., Hovestadt, M., Kao, O., Voss, K.: Peerthing: P2P-based Semantic Resource Discovery. In: Bubak, M., Turała, M., Wiatr, K. (eds.) The 5th Cracow Grid Workshop, pp. 32–38. Academic Computer Center CYFRONET AGH (2005)
4. Funika, W., Kupisz, M., Koperek, P.: Towards Autonomic Semantic-based Management of Distributed Applications. Computer Science 11, 51–63 (2010)
5. Gaia, T., Rizzi, V., Maurizio, M., Lorenzino, V., Paolo, B.: Enabling Information Gathering Patterns for Emergency Response with the OpenKnowledge System. Computing and Informatics 29(4), 537–555 (2010)
6. Gong, L.: Industry Report: JXTA: A Network Programming Environment. IEEE Internet Computing 5(3), 88 (2001)
7. Haase, P., Schnizler, B., Broekstra, J., Ehrig, M., van Harmelen, F., Menken, M., Mika, P., Plechawski, M., Pyszlak, P., Siebes, R., Staab, S., Tempich, C.: Bibster a semantics-based bibliographic Peer-to-Peer system. Web Semantics: Science, Services and Agents on the World Wide Web 2(1), 99–103 (2004)
8. Kaoudi, Z., Miliaraki, I., Koubarakis, M.: RDFS Reasoning and Query Answering on Top of DHTs. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 499–516. Springer, Heidelberg (2008)

9. Kryza, B., Dutka, L., Slota, R., Kitowski, J.: Dynamic VO Establishment in Distributed Heterogeneous Business Environments. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009, Part II. LNCS, vol. 5545, pp. 709–718. Springer, Heidelberg (2009)
10. Kryza, B., Pieczykolan, J., Kitowski, J.: Grid Organizational Memory: A Versatile Solution for Ontology Management in the Grid. In: e-Science, p. 16. IEEE Computer Society (2006)
11. Kryza, B., Slota, R., Majewska, M., Pieczykolan, J., Kitowski, J.: Grid Organizational Memory - Provision of a High-Level Grid Abstraction Layer Supported by Ontology Alignment. Future Generation Comp. Syst. 23(3), 348–358 (2007)
12. Liang, J., Kumar, R., Ross, K.W.: The Fasttrack overlay: A measurement study. Computer Networks 50(6), 842–858 (2006)
13. McBride, B.: Jena: A Semantic Web Toolkit. IEEE Internet Computing 6(6), 55–59 (2002)
14. Mylka, A., Swiderska, A., Kryza, B., Kitowski, J.: Integration of heterogeneous data sources into an ontological knowledge base. Computing and Informatics (in Press, 2012)
15. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: Edutella: a P2P networking infrastructure based on RDF. In: Proc. 11th International World Wide Web Conference, pp. 604–615 (2002)
16. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: ATEC 2004: Proceedings of the Annual Conference on USENIX Annual Technical Conference, p. 10. USENIX Association, Berkeley (2004)
17. Ripeanu, M.: Peer-to-Peer Architecture Case Study: Gnutella Network. In: Graham, R.L., Shahmehri, N. (eds.) Peer-to-Peer Computing, pp. 99–100. IEEE Computer Society (2001)
18. Rowstron, A., 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, pp. 329–350. Springer, Heidelberg (2001)
19. Serafini, L., Tamilin, A.: Drago: Distributed Reasoning Architecture for the Semantic Web. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 361–376. Springer, Heidelberg (2005)
20. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
21. Stelmach, M., Kryza, B., Slota, R., Kitowski, J.: Distributed Contract Negotiation System for Virtual Organizations. Procedia CS 4, 2206–2215 (2011)
22. Tao, Y., Jin, H., Wu, S., Shi, X.: Scalable DHT and ontology-based information service for large-scale grids. Future Generation Comp. Syst. 26(5), 729–739 (2010)
23. Team, S.J.E.: JSRDI: JXTA Shared Resource Distributed Index Design Plan (2002)
24. Tsoumakos, D., Roussopoulos, N.: A Comparison of Peer-to-Peer Search Methods. In: Christophides, V., Freire, J. (eds.) WebDB, pp. 61–66 (2003)
25. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable Distributed Reasoning Using MapReduce. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCs, vol. 5823, pp. 634–649. Springer, Heidelberg (2009)
26. Wei Kang, H.Y., Takahata, K., Shibata, Y.: Large Scale Disaster Information System Based on P2P Overlay Network. In: Barolli, L., Xhafa, F., Vitabile, S., Hsu, H.H. (eds.) CISIS, pp. 261–266. IEEE Computer Society (2010)

Balancing the Communications and Computations in Parallel FEM Simulations on Unstructured Grids

Nikola Kosturski, Svetozar Margenov, and Yavor Vutov

Institute of Information and Communication Technologies,
Bulgarian Academy of Sciences

Abstract. We consider large scale finite element modeling on 3D unstructured grids. Large scale problems imply the use of parallel hardware and software. In general, the computational process on unstructured grids includes: mesh generation, mesh partitioning, optional mesh refinement, discretization, and the solution. The impact of the domain partitioning strategy on the performance of the discretization and solution stages is studied.

Our investigations are focused on the Blue Gene/P massively parallel computer. The mapping of the communications to the underlying 3D tours interconnect topology is considered as well.

As a sample problem, we consider the simulation of the thermal and electrical processes, involved in the radio-frequency (RF) ablation procedure. RF ablation is a low invasive technique for the treatment of hepatic tumors, utilizing AC current to destroy the tumor cells by heating.

1 Introduction

Finite element method (FEM) on unstructured grids has proven to be an indispensable tool in computer modelling. Large scale simulations and complex models require parallel computing. This work is focused on optimizing the performance of parallel simulations.

We use state of the art parallel computer – IBM Blue Gene/P with a state of the art linear solver – BoomerAMG multigrid method[4] from the Hypre library[13]. Our intention was to study the influence of the mesh partitioning on the performance of the entire computational process. We compare two different partitioning libraries: ParMETIS and PT-Scotch. We also, to improve performance, try to map the communication pattern of our programs to the underlying 3D torus interconnect.

Our investigations are done while performing parallel simulation of the radio-frequency (RF) ablation. This is a hepatic tumor treatment technique which uses AC current from an electrode with a complex shape (see Figure 1, a).

The paper is organized as follows: In Section 2, we describe the problem we solve. In Section 3 we discuss and compare the used partitioners. Section 4 contains times from the parallel experiments with and without the mapping of the communications and we finish with a conclusion.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 211–220, 2012.

2 Radio-Frequency Tumor Ablation

Our test problem is numerical simulation of radio-frequency (RF) tumor ablation. RF ablation is an alternative, low invasive technique for the treatment of hepatic tumors, utilizing AC current to destroy the tumor cells by heating ([7,8]). The destruction of the cells occurs at temperatures of 45°C–50°C. The procedure is relatively safe, as it does not require open surgery.

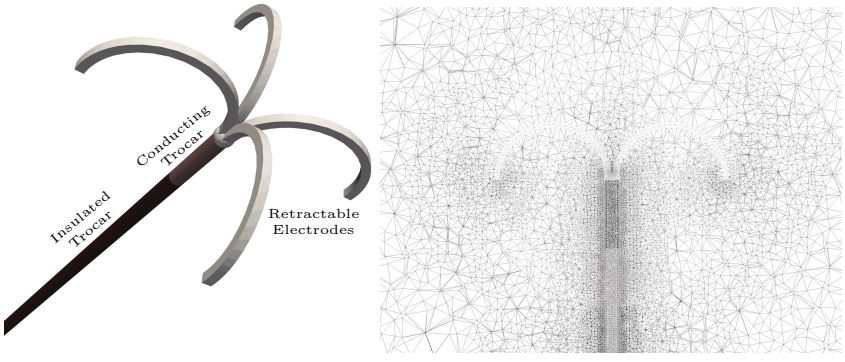


Fig. 1. a) The structure of a fully deployed RF probe; b) Sample mesh: cross-section, different gray levels are used for different materials

The considered RF probe is illustrated on Fig. 1. a) It consists of a stainless steel trocar with four nickel-titanium retractable electrodes. Polyurethane is used to insulate the trocar. The RF ablation procedure starts by placing the straight RF probe inside the tumor. The surgeon performs this under computer tomography (CT) or ultrasound guidance. Once the probe is in place, the electrodes are deployed and RF current is initiated. Both the surfaces areas of the uninsulated part of the trocar and the electrodes conduct RF current.

The human liver has a very complex structure, composed of materials with unique thermal and electrical properties. There are three types of blood vessels with different sizes and flow velocities. Here, we consider a simplified test problem, where the liver consists of homogeneous hepatic tissue and blood vessels.

The RF ablation procedure destroys the unwanted tissue by heating. The heat is dissipated by the electric current flowing through a conductor. The bio-heat time-dependent partial differential equation [7,8]

$$\rho c \frac{\partial T}{\partial t} = \nabla \cdot k \nabla T + J \cdot E - h_{bl}(T - T_{bl}) \tag{1}$$

is used to model the heating process during the RF ablation. The term $J \cdot E$ in (1) represents the thermal energy arising from the current flow and the term $h_{bl}(T - T_{bl})$ accounts for the heat loss due to blood perfusion.

The following initial and boundary conditions are applied

$$\begin{aligned} T &= 37°\text{C when } t = 0 \text{ at } \Omega, \\ T &= 37°\text{C when } t \geq 0 \text{ at } \partial\Omega. \end{aligned} \tag{2}$$

The following notations are used in (1) and (2):

- Ω – the entire domain of the model;
- $\partial\Omega$ – the boundary of the domain;
- ρ – density (kg/m^3);
- c – specific heat (J/kg K);
- k – thermal conductivity (W/m K);
- J – current density (A/m);
- E – electric field intensity (V/m);
- T_{bl} – blood temperature (37°C);
- w_{bl} – blood perfusion (1/s);
- $h_{bl} = \rho_{bl} c_{bl} w_{bl}$ – convective heat transfer coefficient accounting for the blood perfusion in the model.

The bio-heat problem is solved in two steps. The first step is finding the potential distribution V of the current flow. With the considered RF probe design, the current is flowing from the conducting electrodes to a dispersive electrode on the patient's body. The electrical flow is modeled by the Laplace equation

$$\nabla \cdot \sigma \nabla V = 0, \tag{3}$$

with boundary conditions

$$\begin{aligned} V &= 0 \text{ at } \partial\Omega, \\ V &= V_0 \text{ at } \partial\Omega_{el}. \end{aligned}$$

The following notations are used in the above equations:

- V – potential distribution in Ω;
- σ – electric conductivity (S/m);
- V_0 – applied RF voltage;
- $\partial\Omega_{el}$ – surface of the conducting part of the RF probe.

After determining the potential distribution, the electric field intensity can be computed from

$$E = -\nabla V,$$

and the current density from

$$J = \sigma E.$$

The second step is to solve the heat transfer equation (1) using the heat source $J \cdot E$ obtained in the first step.

For the numerical solution of both of the above discussed steps of the simulation the Finite Element Method (FEM) in space is used [10]. Linear conforming elements are used. To solve the bio-heat equation, after the space discretization,

backward Euler scheme is used [11]. There, the time derivative is discretized via finite differences. For a description of the discretization of the problem (3) see [12].

Let us focus on the discrete formulation of the bio-heat equation. Let us denote with K and M the stiffness and mass matrices from the finite element discretization of (1). They can be written as

$$K = \left[\int_{\Omega} k \nabla \Phi_i \cdot \nabla \Phi_j d\mathbf{x} \right]_{i,j=1}^{N},$$

$$M = \left[\int_{\Omega} \rho c \Phi_i \Phi_j d\mathbf{x} \right]_{i,j=1}^{N}.$$

Let us also denote with Ω_{bl} the subdomain of Ω occupied by blood vessels and with M_{bl} the matrix

$$M_{bl} = \left[\int_{\Omega} \delta_{bl} h_{bl} \Phi_i \Phi_j d\mathbf{x} \right]_{i,j=1}^{N},$$

where

$$\delta_{bl}(x) = \begin{cases} 1 \text{ for } x \in \Omega_{bl}, \\ 0 \text{ for } x \in \Omega \setminus \Omega_{bl}. \end{cases}$$

Than, the parabolic equation (1) can be written in matrix form as:

$$M \frac{\partial T}{\partial t} + (K + M_{bl})T = F + M_{bl} T_{bl}. \tag{4}$$

If we denote with τ the time-step, with T^{n+1} the solution at the current time level, and with T^n the solution at the previous time level and approximate the time derivative in (4) we obtain the following system of linear algebraic equations for the nodal values of T^{n+1}

$$(M + \tau(K + M_{bl}))T^{n+1} = MT^n + \tau(F + M_{bl} T_{bl}). \tag{5}$$

In table 1 are given the material properties, taken from [7]. The blood perfusion coefficient $w_{bl} = 6.4 \times 10^{-3}$ 1/s. For the test simulations, a RF voltage of 15 V is applied for a duration of 8 minutes. A time step of $\tau = 10$ s is used.

Table 1. Thermal and Electrical Properties of the Materials

Material	ρ (kg/m^3)	c (J/kg K)	k (W/m K)	σ (S/m)
Ni-Ti	6 450	840	18	1×10^8
Stainless steel	21 500	132	71	4×10^8
Liver	1 060	3 600	0.512	0.333
Blood	1 000	4 180	0.543	0.667
Polyurethane	70	1 045	0.026	10^{-5}

3 Partitioning Methods

Our software tools are written in C++, using MPI for the parallelization. Although we use external libraries for graph partitioning, and for the solution, plenty of nontrivial gluing code was required.

The first stage in the computational process is the mesh generation. Here Netgen package is used [14]. Currently, we use computer generated model of the liver and blood vessels. The RF ablation probe is inserted in the model. Then the geometric data is fed to the generator. In Fig. 1. b) is depicted a cross-section from a sample mesh. The generator output consist in three parts: list of coordinates of the vertices, list of tetrahedrons with assigned materials, and a list of boundaries with assigned boundary conditions.

The next stage is to partition the computational domain among processors. Our intent was to investigate the applicability of two graph partitioning libraries: ParMETIS [2,3] and PT-Scotch [5,6]. To use the graph partitioning routines in both cases we first calculate the dual graph of the mesh using the routine `ParMETIS_V3_Mesh2Dual`. Both libraries require for performance and scalability reasons the initial data to be distributed (fairly) among the processors. This is done by our toolchain as the mesh is read.

The routine `ParMETIS_V3_PartMeshKway` is used for the graph partitioning with ParMETIS. This call computes graph partitioning minimizing the number of cut edges. The result is a part (processor) number, assigned to each tetrahedron. ParMETIS uses parallel version of multilevel k-way partitioning algorithm described in [1].

PT-Scotch library computes graph partitioning via recursive bipartitioning algorithm. PT-Scotch contains ParMETIS compatibility layer, so we use the very same `ParMETIS_V3_PartMeshKway` function from it.

After the partitioning of the elements some postprocessing is required. This includes: distribution of the elements to their processors, determining the partitioning of vertices which are on the interfaces between the processors, distribution of vertex data, distribution of boundary condition data, and node renumbering, which is required for the parallel solver. We assign a shared vertex to a processor with lower number of previously assigned vertices.

Before giving experimental results, let us describe the Blue Gene/P parallel computer. It consist of racks with 1024 PowerPC 450 based compute nodes each. Each node has four computing cores and 2 GB RAM. The nodes are interconnected with several networks. The important ones from the computational point of view are: Tree network for global broadcast, gather, and reduce operations with a bandwidth of 1.7GB/s, 3D torus network for point to point communications, and a separate global interrupt network, used for barriers. In the torus network each node is connected to six other nodes with bidirectional links, each with bandwidth of 0.85GB/s. Torus network is available only when using multiplies of 512 nodes. For smaller number of nodes only 3D mesh interconnect is possible. We were using the machine in so call virtual node mode(VN), in which different MPI rank is assign to each of the computing cores. This is the mode

Table 2. Mesh partitioning

mesh	N_{proc}	Elements avg	Elements $\max^{ParMetis}$	Elements $\max^{PT-Scotch}$	Vertices avg	Vertices $\max^{ParMetis}$	Vertices $\max^{PT-Scotch}$
M0	128	110 453	114 606	110 885	18 561	21 672	20 937
M0	256	55 226	57 494	55 507	9 280	11 363	10 812
M0	512	27 613	28 736	27 765	4 640	5 857	5618
M0	1 024	13 806	14 364	13 902	2 320	3 073	2 968
M1	128	883 627	922 323	888 266	147 654	157 272	151 698
M1	256	441 813	460 387	444 112	73 827	80 836	77 136
M1	512	220 906	230 192	222 074	36 913	41 944	39 366
M1	1 024	110 453	115 055	111 102	18 456	21 796	21 118
M2	1 024	883 627	920 329	888 327	147 405	158 601	154 454
M2	2 048	441 813	461 092	444 934	73 702	81 441	78 815
M2	4 096	220 906	230 500	222 279	36 851	42 419	40 492

to use the entire power of the system, for pure MPI programs (without shared memory parallelism). We were allowed to use up to 1024 computing nodes – 4096 cores, further called processors.

The computation volume for the discretization is proportional to the number of elements in each processor. The computation volume for the solver is proportional to the number of vertices in each processor. Because of the global synchronizations which present in both processes, parallel times will be governed by the maximum number of elements and vertices per processor. In the table 2 these numbers are shown, compared to the averages. Three meshes had been partitioned: M0, M1, and M2. Meshes M1 and M2 are obtained from uniform refinement of mesh M0 once and twice. Let us note that the finest mesh M2 has about 9.0×10^8 elements and about 1.5×10^8 vertices. We expect from a good partitioner per processor maximums to be as close to the averages as possible. Results from partitions on different number of processors are shown.

We can clearly see that PT-Scotch produces better partitions. Both the maximum number of elements per processor and the maximum number of vertices per processor are closer to the optimal values.

To give an idea about the communication pattern resulting from the partitionings, we have shown some info about the connections in table 3. We call two processors connected if they share at least one vertex in their elements. Connected processors must exchange data during the discretization and the solution processes. The minimum – C_{min}, average – C_{avg}, and maximum – C_{min} number of connection for an processor are shown. The other column N_{max} is defined as follows:

$$N_{max} = \max_{0<i<N_{proc}} \sum_{j=1}^{N_{proc}} N_{i,j}, \qquad (6)$$
$$N_{i,j} = \max(S_{i,j}, R_{i,j}),$$

where N_{proc} is the number of processors, $S_{i,j}$ is the number of values sent from processor i to processor j and $R_{i,j}$ is the number of received values. In other

Table 3. Communication volume

		ParMETIS					PT-Scotch				
mesh	N_{proc}	C_{min}	C_{avg}	C_{max}	N_{max}	t[s]	C_{min}	C_{avg}	C_{max}	N_{max}	t[s]
M0	128	4	12	28	73 108	2.96	5	12	22	63 728	39.0
M0	256	4	13	26	47 392	1.59	3	13	25	39 020	35.9
M0	512	5	14	26	30 596	1.03	4	14	27	25 714	33.9
M0	1 024	5	14	27	19 376	0.87	4	14	26	16 652	32.8
M1	128	5	12	25	225 788	23.0	5	13	23	172 096	125
M1	256	7	14	29	155 912	11.5	3	13	27	118 948	92.7
M1	512	5	14	29	103 508	5.94	5	14	26	78 774	72.5
M1	1 024	5	15	32	80 360	3.46	5	14	25	67 392	61.11
M2	1 024	7	15	30	271 951	25.1	4	15	27	206 942	192
M2	2 048	5	16	30	162 946	13.4	5	15	29	137 587	138
M2	4 096	4	16	30	106 734	8.01	4	15	28	97 510	111

words N_{max} is the maximum amount of data a processor communicates. We also give the time t for the partitioning in seconds. We see in both cases similar number of connections. The communication volume N_{max} is lower for PT-Scotch. Although PT-Scotch produces better partitions, we see that it does not scale. Its run times are several times longer than those of ParMETIS.

The matrices of the linear systems are ill-conditioned and large. Since they are symmetric and positive definite, we use the PCG [9] method, with a BoomerAMG as a preconditioner. The settings for the BoomerAMG preconditioner were carefully tuned for maximum scalability. The selected coarsening algorithm is *Falgout-CLJP. Modified classical interpolation* is applied. The selected relaxation method is *hybrid symmetric Gauss-Seidel or SSOR*. To decrease the operator and grid complexities two levels of aggressive coarsening are used. Smaller operator and grid complexities are lead to faster iterations and reduced memory requirements. This is essential on BlueGene/P, as each processor has only 512MB of RAM. The downside is that this affect the convergence rate of the solver.

In table 4 are shown parallel times for the discretization – T_{disc} and the solution of the linear system – T_{solve} for the bioheat problem 1. The matrix $A = M + \tau(K + M_{bl})$ from (5) is assembled only once on the first time step and not varied after that. The corresponding AMG preconditioner is also constructed only on the first time step. We see that the discretization is faster and the linear solver performs better for the partitions produced by PT-Scotch. The discretization part scales nicely. We see that the linear solver fails to scale for (relatively) small problems, and beyond 2 048 processors.

We asked ourselves the question "can we do better?" We tried to use the static graph mapping capabilities of the Scotch library. A mapping is called static if it is computed prior to the execution of the program. Static mapping is NP-complete in the general case [5]. Scotch library uses suboptimal method of Dual recursive bipartitioning. The parallel program to be mapped onto the target architecture

Table 4. Computation times

		ParMETIS		PT-Scotch	
mesh	N_{proc}	$T_{disc}[s]$	$T_{solve}[s]$	$T_{disc}[s]$	$T_{solve}[s]$
M0	128	515	54.6	471	51.2
M0	256	247	34.6	249	29.7
M0	512	138	23.8	133	23.4
M0	1 024	72.8	22.7	69.9	21.3
M1	128	4 225	332	4 007	319
M1	256	2 160	173	2 101	167
M1	512	1 196	101	1 131	97.9
M1	1 024	608	75.2	574	69.0
M2	1 024	5 381	356	4 856	337
M2	2 048	2 742	224	2 364	211
M2	4 096	1 322	204	1 215	207

is modeled by a weighted unoriented graph S called source graph or process graph, the vertices of which represent the processes of the parallel program, and the edges of which are the connections between communicating processes. The target machine onto which is mapped the parallel program is also modeled by a weighted unoriented graph T called target graph or architecture graph. The algorithm starts with the set of processors in T, also called domain, which is associated the set of all the processes in S to map. At each step, the algorithm bipartitions a yet unprocessed domain into two disjoint subdomains, and calls a graph bipartitioning algorithm to split the subset of processes.

In our case the we use $N_{i,j}$ from (6) as edge weights for S. We set the weights on the edges of T to be the number of hops between a pair of nodes plus one (there is a non-zero weight requirement in Scotch). We take into account that the mesh topology is mesh up to 1024 processors and torus in other two cases. The mapping is computed with the tool *gmap*, with an option which does not allow to assign two processes to one processor. In our solver, we construct an MPI communicator with reordered ranks to match the mapping. We use that communicator for the computations. We performed the same tests, bur with mapped communications. The results are shown in table 5. To ease the comparison only the ratios $T^{mapped}/T^{non-mapped}$ are shown in percents, where $T^{non-mapped}$ are the corresponding values in table 4.

In all cases the performance of the linear solver is improved. Although the improvements are small – under 2%, we can say that that the mapping has a positive effect. Things differ for the discretization times. For the ParMETIS partitionings, there is a small improvement in most cases, expect three. But for the PT-Scotch partitions things got worse in all cases. We expected that because of the algorithm PT-Scotch uses and because of the nature of the interconnect, its partitions were already properly mapped. But the slower run times can only mean that there is an overhead in using communicator different from `MPI_COMM_WORLD`.

Table 5. Computation mapping

		ParMETIS		PT-Scotch	
mesh	N_{proc}	T_{disc}[%]	T_{solve}[%]	T_{disc}[%]	T_{solve}[%]
M0	128	96.1	99.7	101.4	99.4
M0	256	98.1	100.0	102.2	99.6
M0	512	99.8	98.9	103.6	99.2
M0	1 024	100.1	99.7	103.6	99.7
M1	128	99.6	99.4	102.5	99.7
M1	256	99.1	99.8	101.6	99.2
M1	512	99.5	99.1	101.8	98.2
M1	1 024	101.9	99.1	104.1	98.1
M2	1 024	100.7	98.6	104.3	98.7
M2	2 048	98.7	99.7	104.2	99.7
M2	4 096	99.5	98.9	107.2	99.0

4 Conclusion and Future Work

In this work we compared the impact two graph partitioning libraries on the performance of parallel FEM simulations. We saw that partitioning quality has direct and non-negligible influence on the performance.

We used a tool for static graph mapping to optimize the communications on massively parallel computer. The results show that we could gain by this kind of optimizations, especially on problems with heavy communication loads.

Our future plans include tuning the parameters. Scotch library offers many possibilities to influence its partitioning results. This is also true for the static mapping. We also expect a parallel version of the routine `METIS_PartGraphVKway` from the Metis library, which directly minimizes total communication volume.

We intent to use process remapping method, different from currently used MPI communicator with reordered ranks.

Acknowledgments. This work is partly supported by the Bulgarian NSF Grants DCVP 02/1 and DPRP7RP02/13. We also kindly acknowledge the support of the Bulgarian Supercomputing Center for the access to the IBM Blue Gene/P supercomputer.

References

1. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and DistributedComputing 48(1) (1998)
2. Karypis, G., Kumar, V.: A coarse-grain parallel multilevel k-way partitioning algorithm. In: Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing (1997)
3. ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering, `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`

4. Henson, V.E., Yang, U.M.: BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics 41(1), 155–177 (2002)
5. Pellegrini, F., Roman, J.: Experimental analysis of the dual recursive bipartitioning algorithm for static mapping. Research Report, LaBRI, Universit e Bordeaux I (August 1996), http://www.labri.fr/~pelegrin/papers/scotch_expanalysis.ps.gz
6. Scotch and PT-Scotch: Software package and libraries for sequential and parallel graph partitioning, static mapping, and sparse matrix block ordering, and sequential mesh and hypergraph partitioning, http://www.labri.fr/perso/pelegrin/scotch/
7. Tungjitkusolmun, S., Staelin, S.T., Haemmerich, D., Tsai, J.Z., Cao, H., Webster, J.G., Lee, F.T., Mahvi, D.M., Vorperian, V.R.: Three-dimensional finite-element analyses for radio-frequency hepatic tumor ablation. IEEE Transactions on Biomedical Engineering 49(1), 3–9 (2002)
8. Tungjitkusolmun, S., Woo, E.J., Cao, H., Tsai, J.Z., Vorperian, V.R., Webster, J.G.: Thermal-electrical finite element modelling for radio frequency cardiac ablation: Effects of changes in myocardial properties. Medical and Biological Engineering and Computing 38(5), 562–568 (2000)
9. Axelsson, O.: Iterative Solution Methods. Cambridge University Press (1996)
10. Brenner, S., Scott, L.: The mathematical theory of finite element methods. Texts in Applied Mathematics, vol. 15. Springer, Heidelberg (1994)
11. Hairer, E., Norsett, S.P., Wanner, G.: Solving ordinary differential equations I, II. Springer Series in Comp. Math. (2002)
12. Kosturski, N., Margenov, S.: Supercomputer Simulation of Radio-Frequency Hepatic Tumor Ablation, AMiTaNS 2010 Proceedings. AIP CP, vol. 1301, pp. 486–493 (2010)
13. Lawrence Livermore National Laboratory, Scalable Linear Solvers Project, http://www.llnl.gov/CASC/linear_solvers/
14. Netgen Mesh Generator, http://sourceforge.net/apps/mediawiki/netgen-mesher/

Scalable Quasineutral Solver for Gyrokinetic Simulation

Guillaume Latu[1,2], Virginie Grandgirard[1],
Nicolas Crouseilles[2], and Guilhem Dif-Pradalier[3]

[1] CEA Cadarache, 13108 Saint-Paul-les-Durance, France
[2] INRIA Nancy-Grand Est, 54600 Villers-lès-Nancy, France
[3] UCSD, La Jolla, CA 92093, USA

Abstract. Modeling turbulent transport is a major goal in order to predict confinement issues in a tokamak plasma. The gyrokinetic framework considers a computational domain in five dimensions to look at kinetic issues in a plasma. Gyrokinetic simulations lead to huge computational needs. Up to now, the semi-Lagrangian code GYSELA performed large simulations using a few thousands of cores. The work proposed here improves GYSELA onto two points: memory scalability and execution time. The new solution allows the GYSELA code to scale well up to 64k cores.

Keywords: Quasineutrality solver, Gyrokinetics, MPI, OpenMP.

1 Introduction

To have access to a kinetic description of the plasma dynamics inside a tokamak, one usually needs to solve the Vlasov equation nonlinearly coupled to Maxwell equations. Then, a parallel code addressing the issue of modelling tokamak plasma turbulence needs to couple a 5D parallel Vlasov solver with a 3D parallel field solver (Maxwell). On the first hand, a Vlasov solver moves the plasma particles forward in time. On the other hand, the field solver gives the electromagnetic fields generated by a given particles setting in phase space.

Computational resources available nowadays has allowed the development of several petascale codes based on the well-established gyrokinetic framework. In the last decade, the simulation of turbulent fusion plasmas in Tokamak devices has involved a growing number of people coming from the applied mathematics and parallel computing fields [5,1,9].

In this paper, we focus on the study of the field solver embedded in the GYSELA gyrokinetic code. A new algorithm is presented here that has excellent performance up to a few thousands of cores. An adapted communication scheme is introduced to reduce the communication costs for exchanging information between the semi-Lagrangian Vlasov solver and the field solver. From 4k cores to 64k cores, the field solver tends towards an asymptotic time cost. Nevertheless, its small computation cost compared to the other parts, allows GYSELA to get good performance. Relative efficiency has been measured at 78% on 64k cores. In the last sections, performance is evaluated both in term of execution time and memory scalability.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 221–231, 2012.

2 Gyrokinetic Model

2.1 Parallel Solving of Vlasov Equation

Our gyrokinetic model considers as main unknown a distribution function $\bar{f}$ that represents the density of ions at a given phase space position. This function depends on time and on 5 other dimensions. First, 3 dimension in space, r and θ are the polar coordinates in the shortest cross-section of the torus (called poloidal section), while φ refers to the angle in the largest cross-section of the torus. Second, velocity space has two dimensions: $v_\parallel$ being the velocity along the magnetic field lines (one has $v_\parallel = \mathrm{d}\varphi/\mathrm{d}t$), and μ the magnetic moment corresponding to the action variable associated with the gyrophase. The time evolution of the guiding-center 5D gyroaveraged distribution function $\bar{f}_t(r, \theta, \varphi, v_\parallel, \mu)$ is governed by the so-called gyrokinetic equation [4,2] :

$$\frac{\partial \bar{f}}{\partial t} + \frac{\mathrm{d}r}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial r} + \frac{\mathrm{d}\theta}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \theta} + \frac{\mathrm{d}\varphi}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \varphi} + \frac{\mathrm{d}v_\parallel}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial v_\parallel} = 0. \tag{1}$$

In this Vlasov gyrokinetic equation, μ acts as a parameter because it is an adiabatic motion invariant. Let us denote by N_μ the number of μ values, we have N_μ independent Eq. 1 to solve at each time step. The function $\bar{f}$ is periodic along θ and φ. Vanishing perturbations are imposed at the boundaries in the non-periodic directions r and $v_\parallel$.

GYSELA is a global nonlinear electrostatic code which solves the gyrokinetic equations with a semi-Lagrangian method [3,2]. We combine this with a second order in time Strang splitting method. We refer the reader to [6,2] for detailed description of Vlasov solver, we will only recall a few issues concerning the parallel domain decomposition. Large data structures are used in GYSELA: the 5D data $\bar{f}$, and the 3D data as the electric potential Φ. Let N_r, N_θ, N_φ, $\mathrm{N}_{v_\parallel}$ be the number of points in each dimension r, θ, φ, $v_\parallel$.

In the Vlasov solver, we give the responsibility of each value of μ to a given set of MPI processes [6] (a MPI communicator). We fixed that there are always N_μ sets, such as only one μ value is attributed to each communicator. Within each set, a 2D domain decomposition allows us to attribute to each MPI Process a subdomain in (r, θ) dimensions. Thus, a MPI process is then responsible for the storage of the subdomain defined by $\bar{f}(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *, v_\parallel = *, \mu = \mu_{value})$. The parallel decomposition is initially set up knowing local values $i_{start}, i_{end}, j_{start}, j_{end}, \mu_{value}$. They are derived from a classical block decomposition of the r domain into $\mathrm{p_r}$ pieces, and of the θ domain into p_θ subdomains. The numbers of MPI processes used during one run is equal to $\mathrm{p_r} \times \mathrm{p}_\theta \times \mathrm{N}_\mu$. The OpenMP paradigm is used in addition to MPI (#T threads in each MPI process).

2.2 Quasineutrality Equation

The quasineutrality equation and parallel Ampere's law close the self-consistent gyrokinetic Vlasov-Maxwell system. However, in an electrostatic code, the field

solver reduces to the numerical solving of a Poisson-like equation [4]. In tokamak configurations, the plasma quasineutrality (denoted QN) approximation is currently assumed [2]. Electron inertia is ignored, which means that an adiabatic response of electrons is supposed. We define the operator $\nabla_\perp = (\partial_r, \frac{1}{r}\partial_\theta)$. We note n_0 the equilibrium density, B_0 the magnetic field at the magnetic axis and $T_e(r)$ the electronic temperature. We have also $B(r,\theta)$ the magnetic field, J_0 the Bessel function of first order and $k_\perp$ the transverse component of the wave vector. Hence, the QN equation can be written in dimensionless variables

$$-\frac{1}{n_0(r)}\nabla_\perp \cdot \left[\frac{n_0(r)}{B_0}\nabla_\perp \Phi(r,\theta,\varphi)\right] + \frac{1}{T_e(r)}\left[\Phi(r,\theta,\varphi) - \langle\Phi\rangle_{\theta,\varphi}(r)\right] = \tilde{\rho}(r,\theta,\varphi) \qquad (2)$$

where $\tilde{\rho}$ is defined by

$$\tilde{\rho}(r,\theta,\varphi) = \frac{2\pi}{n_0(r)}\int B(r,\theta)d\mu \int dv_\parallel J_0(k_\perp\sqrt{2\mu})(\bar{f} - \bar{f}_{eq})(r,\theta,\varphi,v_\parallel,\mu). \qquad (3)$$

with $\bar{f}_{eq}$ representing local ion Maxwellian equilibrium, and $\langle\rangle_{\theta,\varphi}$ the average onto the variables θ, φ. The presence of the non-local term $\langle\Phi\rangle_{\theta,\varphi}(r)$ couples (θ,φ) dimensions and penalizes the parallelization. We employ a solution based on FFT to overcome this problem. But this method is valid only in polar coordinates and not adapted to all geometries [8]. The QN solver includes two parts. First, the function $\tilde{\rho}$ is derived taking as input function $\bar{f}$ that comes from the Vlasov solver. In Eq. (3) specific methods (*e.g.* see [7]) are used to evaluate the gyroaverage operator J_0 on $(\bar{f} - \bar{f_{eq}})$. Second, the 3D electric potential Φ is derived. The present solution retains components of [7], while reducing parallel overheads.

In the following, we will refer to several data as *3D field data*. They are produced and distributed over the parallel machine just after the QN solver. These field data sets, namely: Φ, $\frac{\partial J_0(k_\perp\sqrt{2\mu})\Phi}{\partial r}$, $\frac{\partial J_0(k_\perp\sqrt{2\mu})\Phi}{\partial \theta}$, $\frac{\partial J_0(k_\perp\sqrt{2\mu})\Phi}{\partial \varphi}$, are distributed on processes in a way that exclusively depends on the parallel domain decomposition chosen in the Vlasov solver. Indeed, they are inputs for the Vlasov Eq. (1), and they play a major role in terms $\frac{\mathrm{d}r}{\mathrm{d}t}$, $\frac{\mathrm{d}\theta}{\mathrm{d}t}$, $\frac{\mathrm{d}\varphi}{\mathrm{d}t}$, $\frac{\mathrm{d}v_\parallel}{\mathrm{d}t}$ not detailed here. So we need that the subdomain owned by each MPI process has the form $(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *)$ for the 3D field data, as we shall see.

3 Scalable Algorithm for the QN Solver

3.1 1D Fourier Transforms Method

The method that follows considers only 1D FFTs in θ dimension and uncouples computations in φ direction. The equation (2) averaged on (θ,φ) gives :

$$-\frac{\partial^2\langle\Phi\rangle_{\theta,\varphi}(r)}{\partial r^2} - [\frac{1}{r} + \frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\langle\Phi\rangle_{\theta,\varphi}(r)}{\partial r} = \langle\tilde{\rho}\rangle_{\theta,\varphi}(r) \qquad (4)$$

A Fourier transform in θ direction leads to:

$$\Phi(r,\theta,\varphi) = \textstyle\sum_m \hat{\Phi}^m(r,\varphi)e^{i\,m\,\theta}, \quad \tilde{\rho}(r,\theta,\varphi) = \textstyle\sum_m \hat{\rho}^m(r,\varphi)e^{i\,m\,\theta}$$

The equation (2) could be rewritten as:

$$\text{for } m>0: \quad -\frac{\partial^2\hat{\Phi}^m(r,\varphi)}{\partial r^2}-[\frac{1}{r}+\frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\hat{\Phi}^m(r,\varphi)}{\partial r}+\frac{m^2}{r^2}\hat{\Phi}^m(r,\varphi)+\frac{\hat{\Phi}^m(r,\varphi)}{T_e(r)}=\hat{\rho}^m(r,\varphi) \quad (5)$$

$$\text{for } m=0: \quad \frac{\partial^2\langle\Phi\rangle_\theta(r,\varphi)}{\partial r^2}-[\frac{1}{r}+\frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\langle\Phi\rangle_\theta(r,\varphi)}{\partial r}+\frac{\langle\Phi\rangle_\theta(r,\varphi)-\langle\Phi\rangle_{\theta,\varphi}(r)}{T_e(r)}=\langle\tilde{\rho}\rangle_\theta(r,\varphi) \quad (6)$$

The equation (4) allows one to directly find out the value of $\langle\Phi\rangle_{\theta,\varphi}(r)$ from the input data $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r)$. Let us define the function $\Upsilon(r,\theta,\varphi)$ as $\Phi(r,\theta,\varphi)-\langle\Phi\rangle_{\theta,\varphi}(r)$. Substracting equation (4) to equation (6) leads to

$$-\frac{\partial^2\langle\Upsilon\rangle_\theta(r,\varphi)}{\partial r^2}-[\frac{1}{r}+\frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\langle\Upsilon\rangle_\theta(r,\varphi)}{\partial r}+\frac{\langle\Upsilon\rangle_\theta(r,\varphi)}{T_e(r)}=\langle\tilde{\rho}\rangle_\theta(r,\varphi)-\langle\rho\rangle_{\theta,\varphi}(r) \quad (7)$$

Let us notice that $\hat{\Phi}^0(r,\varphi)=\langle\Upsilon\rangle_\theta(r,\varphi)+\langle\Phi\rangle_{\theta,\varphi}(r)$. So, the solving of equations (4) and (7) allows one to compute $\langle\Phi\rangle_{\theta,\varphi}(r)$,$\langle\Upsilon\rangle_\theta(r,\varphi)$ and $\hat{\Phi}^0(r,\varphi)$ from the quantities $\langle\tilde{\rho}\rangle_\theta(r,\varphi)$ and $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r)$. Then, Eq. (5) is sufficient to compute $\hat{\Phi}^{m>0}(r,\varphi)$ from $\tilde{\rho}$. The equations are solved using a LU decomposition precomputed once. As φ acts as a parameter in Eq. (5), parallelization is possible over this dimension.

3.2 Data Distribution Issues

In the Vlasov solver and the beginning of QN solver each process knows the values of a subdomain $\bar{f}(r=[i_{start},i_{end}],\theta=[j_{start},j_{end}],\varphi=*,v_\parallel=*,\mu=\mu_{value})$. The field Φ is an output of QN solver that we would like to distribute over the parallel machine. In an earlier work [7], we had simplified the problem of data dependancies, in broadcasting the entire Φ data structure to all processes. Then we computed in each MPI process the derivatives of gyroaveraged electric potential redondantly. Nevertheless, this strategy leads to a bottleneck for large platforms (typically more than 4k cores). Indeed, the broadcast involves growing communication costs along with the number of processes, and the sequential nature of derivatives computation becomes also problematic. These two overheads are unnecessary, even they simplify the implementation of several subroutines and reduces also the complexity of data management. In the presented version, only a subdomain of Φ is sent to each process. Also, a distributed algorithm computes the derivatives of $J_0(k_\perp\sqrt{2\mu})\Phi$, as you will see in Algo. 2.

3.3 Parallel Algorithm Descriptions

The Algo. 1 describes a scalable parallel algorithm of the QN solver. This formulation is not yet valid in toroidal setting, cylindrical geometry is used. It improves previous approaches [7,2] in introducing a better work distribution, and also in reducing the final communication. The main idea of the algorithm is to get $\langle\Phi\rangle_{\theta,\varphi}$ for solving Eq. (7), and then to uncouple computations of $\hat{\Phi}^m$ along φ direction

in the QN solver. Finally, each locally computed $\Phi(r, \theta, \varphi)$ values are sent to process that is responsible for it. The algorithm has parameters: the mappings s, g and q that are detailed in the next subsection.

Algorithm 1. Parallel algorithm for the QN solver

Input : local block
$\bar{f}(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *, v_{\parallel} = *, \mu = \mu_{value})$
(task 1*)*
Computation : $\tilde{\rho}_1$ by integration in $dv_{\parallel}$ of $\bar{f}$ `/* parallel in` μ, r, θ `*/`
Send local data $\tilde{\rho}_1(r = [i_{start}, i_{end}]), \theta = [j_{start}, j_{end}], \varphi = *, \mu = \mu_{value})$
Redistribute $\tilde{\rho}_1$ **/ Synchronization**
Receive block $\tilde{\rho}_1(r = *, \theta = *, \varphi = [s^{\varphi}_{start}, s^{\varphi}_{end}], \mu = [s^{\mu}_{start}, s^{\mu}_{end}])$
(task 2*)*
for $\varphi = [s^{\varphi}_{start}, s^{\varphi}_{end}]$ and $\mu = [s^{\mu}_{start}, s^{\mu}_{end}]$ **do** `/* parallel in` μ, φ `*/`
  Computation : from $\tilde{\rho}_1$ at one φ, compute $\tilde{\rho}_2$ applying $J_0(k_{\perp}\sqrt{2\mu})$ (Fourier transform in θ, Solving of LU systems in r) $\forall \mu \in [s^{\mu}_{start}, s^{\mu}_{end}]$
  Computation : $\tilde{\rho}_3$ for a given φ by integration in $d\mu$ of $\tilde{\rho}_2$
if $[s^{\mu}_{start}, s^{\mu}_{end}] \neq [0, N_{\mu} - 1]$ **then**
  Send local data $\tilde{\rho}_3(r = *, \theta = *, \varphi = [s^{\varphi}_{start}, s^{\varphi}_{end}])$
  Reduce Sum $\tilde{\rho}$ += $\tilde{\rho}_3$ **/ Synchronization**
  Receive summed block $\tilde{\rho}(r = *, \theta = *, \varphi = [g_{start}, g_{end}])$
(task 3*)*
for $\varphi = [g_{start}, g_{end}]$ **do** `/* parallel in` φ `*/`
  Computation : accumulation of $\tilde{\rho}$ values to get $\langle\tilde{\rho}\rangle_{\theta}(r = *, \varphi)$
Send local data $\langle\tilde{\rho}\rangle_{\theta(r=*,\varphi=[g_{start},g_{end}])}$
Broadcast of $\langle\tilde{\rho}\rangle_{\theta}$ **/ Synchronization**
Receive $\langle\tilde{\rho}\rangle_{\theta(r=*,\varphi=*)}$
(task 4*)*
Computation : Solving of LU system to find $\langle\Phi\rangle_{\theta,\varphi}$ from $\langle\tilde{\rho}\rangle_{\theta,\varphi}$, eq. (3)
for $\varphi = [g_{start}, g_{end}]$ **do** `/* parallel in` φ `*/`
  Computation : 1D FFTs of $\tilde{\rho}$ on dimension (θ)
  Computation : Solving of LU systems for $\hat{\Phi}^m$ modes $(\forall m > 0)$, eq. (5)
  Computation : Solving of LU system for $\langle\Upsilon\rangle_{\theta}(r=*,\varphi)$, eq. (7)
  Computation : Adding $\langle\Phi\rangle_{\theta,\varphi}$ to $\langle\Upsilon\rangle_{\theta}(r=*,\varphi)$ gives $\hat{\Phi}^0(r=*,\varphi)$
  Computation : inverse 1D FFTs on $\hat{\Phi}^0$ and $\hat{\Phi}^{m>0}$ to get $\Phi(r=*,\theta=*,\varphi)$
Send local data $\Phi(r = *, \theta = *, \varphi = [g_{start}, g_{end}])$ to the N_{μ} communicators
Broadcast of values / Synchronization
Receive global data $\Phi(r = *, \theta = *, \varphi = [q_{start}, q_{end}])$
Outputs : $\Phi(r = *, \theta = *, \varphi = [q_{start}, q_{end}])$

Algorithm 2. Parallel algorithm to get derivatives of the potential

Input : local block $\Phi(r = *, \theta = *, \varphi = [q_{start}, q_{end}])$
(task 1*)*
for $\varphi = [q_{start}, q_{end}]$ and $\mu = \mu_{value}$ **do** `/* parallel in` μ, φ `*/`
  Computation : $A_0(r = *, \theta = *, \varphi) = J_0(k_{\perp}\sqrt{2\mu})\,\Phi(r = *, \theta = *, \varphi)$
  $A_1(r = *, \theta = *, \varphi) = \frac{\partial A_0(r=*,\theta=*,\varphi)}{\partial r}$
  $A_2(r = *, \theta = *, \varphi) = \frac{\partial A_0(r=*,\theta=*,\varphi)}{\partial \theta}$
Send local data $A_0|A_1|A_2(r = *, \theta = *, \varphi = [q_{start}, q_{end}])$
Redistribute $A_0|A_1|A_2$ **inside** μ **communicator/ Synchronization**
Receive blocks $A_0|A_1|A_2(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *)$
(task 2*)*
for $r = [i_{start}, i_{end}]$, $\theta = [j_{start}, j_{end}]$, $\mu = \mu_{value}$ **do** `/* parallel in` μ, r, θ `*/`
  Computation : $A_3(r, \theta, \varphi = *) = \frac{\partial A_0(r,\theta,\varphi=*)}{\partial \varphi}$
Outputs : $A_1|A_2|A_3(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *)$

The computation sequence is: integrate $\bar{f}$ over $v_{\parallel}$ direction (task 1), compute right-hand side $\tilde{\rho}$ in summing over μ (task 2), perform averages $\langle\tilde{\rho}\rangle_{\theta}(r=*,\varphi=*)$ and $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r=*)$ (task 3), solve QN equation and produces Φ slices (task 4).

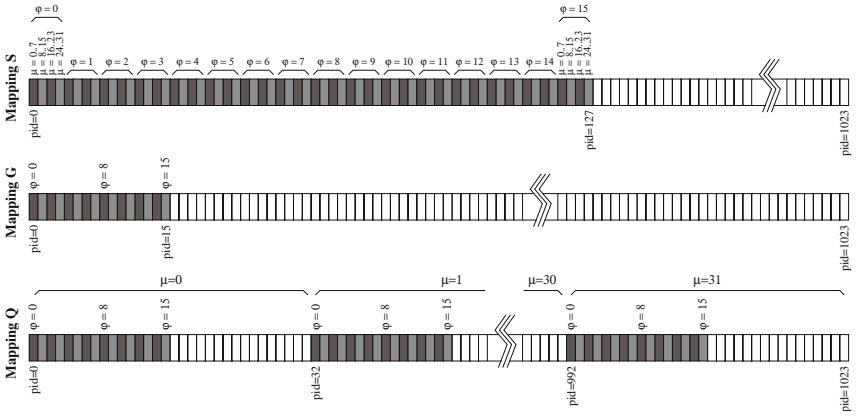


Fig. 1. Mappings of μ, φ ($\mathrm{N}_\mu = 32, \mathrm{N}_\varphi = 16$) on processes, large testbed (#P = 1024)

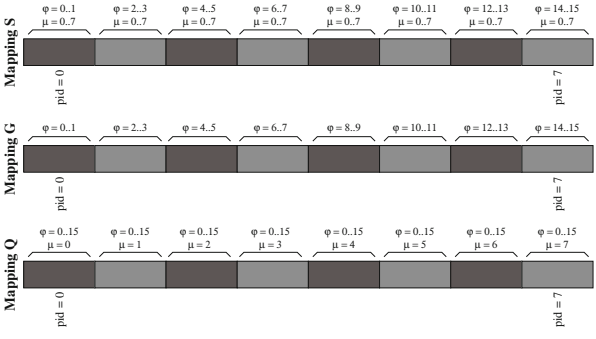


Fig. 2. Mappings of μ, φ ($\mathrm{N}_\mu = 8, \mathrm{N}_\varphi = 16$) on processes, small testbed (#P = 8)

The Algo. 2 follows immediatly the QN solver. It applies the gyroaverage on distributed Φ data and then computes its derivatives along spatial dimensions. These 3D fields are named A_1, A_2, A_3 in the algorithm. They are redistributed in a communication step in order to correctly feed the Vlasov solver. In the task 2, derivatives along φ direction are computed (all values are known along φ).

3.4 Mapping Functions

The two presented algorithms use three different mappings to distribute computations and data on the parallel machine. These mappings concern φ and μ variables and are illustrated in Fig. 1 and 2 for a large testbed and a small one respectively. Let #C be the number of cores used for a simulation run and #P be the number of MPI process. The number of threads #T per MPI process is fixed, so we have #C = #P#T. Each rectangle on the Figures 1 and 2 represents a MPI process. A process filled in dark or light gray has computations to perform,

whereas the white color denotes an idle process. These mappings also implicitly prescribe how communication schemes exchange data during the execution of the two algorithms. We give here a brief description of these mappings:

- **Mapping S** - It defines the ranges $\varphi \in [s^{\varphi}_{start}, s^{\varphi}_{end}]$ and $\mu \in [s^{\mu}_{start}, s^{\mu}_{end}]$. In the task 2 of QN solver, we use this mapping to distribute the computation of the gyroaverage J_0. The maximal parallelism is then obtained whenever each core has at most one gyroaverage operator to apply. We have considered in the example shown in Fig. 1 that each MPI process hosts $\#\mathrm{T} = 8$ threads, so that a process can deal with 8 gyroaveraging simultanously ($s^{\mu}_{end} - s^{\mu}_{start} + 1 = 8$). This distribution is computed in establishing a block distribution of domain $[0, \mathrm{N}_{\mu} - 1] \times [0, \mathrm{N}_{\varphi} - 1]$ over $\#\mathrm{C}$ cores.
- **Mapping G** - It defines the range $[g_{start}, g_{end}]$ for the φ variable. A simple block decomposition is used along φ dimension. For a large number of cores, this distribution gives to processes 0 to $\mathrm{N}_{\varphi} - 1$ the responsability to compute the N_{φ} slices of Φ data structure (task 4 of the QN solver). For a small testbed (see Fig. 2), this mapping is identical to S mapping ($[s^{\mu}_{start}, s^{\mu}_{end}] = [0, N_{\mu} - 1]$). Then, we save the communication step in task 2 of QN solver.
- **Mapping Q** - The mapping Q defines the range $\varphi \in [q_{start}, q_{end}]$ in each MPI process. Inside each μ communicator, it creates a block decomposition along φ dimension. It is designed to carry out the computation of the gyroaverage of Φ, together with the computation of its derivatives (Algo. 2). These calculations depend on the value of μ. It is cost effective to perform them inside μ communicators: we need to only locally redistribute data inside the μ communicator at the end of Algo. 2 in order to prepare the input 3D data fields for the Vlasov solver.

3.5 Communication Costs Analysis

Let us estimate communication costs over all processes associated with the Algo. 1. From line 6 to line 8, a 4D data $\tilde{\rho}_1$ is redistributed (a global transposition that involves all MPI processes). The amount of communication represents the exchange of $\mathrm{N}_r\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\mathrm{N}_{\mu}$ floats. For the lines 18 to 20, the amount of communication is strictly lower to $\mathrm{N}_r\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\mathrm{N}_{\mu}$ floats, and tightly depends on the mapping S. The broadcast of lines 28 to 30 corresponds to a smaller communication cost of $\mathrm{N}_r\mathrm{N}_{\varphi}\min(\#\mathrm{C}, \mathrm{N}_{\varphi})$. The final communication of lines 42-44 sends N_{μ} times each 2D slice of Φ (locally computed). Its costs is near the cost of lines 6-8 with $\mathrm{N}_r\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\mathrm{N}_{\mu}$ floats, but without data transpose (so less memory copies).

In Algo. 2, a communication step transpose three 3D data inside each μ communicator. The overall cost is $3\,\mathrm{N}_r\,\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\,\mathrm{N}_{\mu}$ floats, but only local communications inside μ communicators are realized.

4 Performance Analysis

Timing measurements have been performed on CRAY-XT5 Jaguar machine (DOE). This machine has 18 688 XT5 nodes hosting dual hex-core AMD

Table 1. Time measurements for one call to the QN solver - Small case

Nb. cores	256	1k	4k
Nb. nodes	32	128	512
comp1	2300 ms	580 ms	100 ms
io1	300 ms	160 ms	90 ms
comp2	170 ms	43 ms	13 ms
io2	0 ms	0 ms	8 ms
comp3	0 ms	0 ms	2 ms
io3	17 ms	42 ms	43 ms
comp4	4 ms	3 ms	4 ms
io4	100 ms	40 ms	35 ms
Total time	2900 ms	870 ms	300 ms
Relative eff.	100%	83%	60%

Table 2. Time measurements for one call to the QN solver - Big case

Nb. cores	4k	16k	64k
Nb. nodes	512	2k	8k
comp1	2200 ms	570 ms	95 ms
io1	450 ms	300 ms	470 ms
comp2	320 ms	180 ms	180 ms
io2	30 ms	60 ms	70 ms
comp3	3 ms	2 ms	3 ms
io3	150 ms	140 ms	130 ms
comp4	30 ms	30 ms	30 ms
io4	180 ms	100 ms	65 ms
Total time	3400 ms	1400 ms	1000 ms
Relative eff.	100%	61%	21%

processors, 16 GB of memory. The Table 1 reports timing of the QN solver extracted from GYSELA runs. N_μ remains constant while $\mathrm{p_r}$ and p_θ are increased. A small case was run from 256 cores to 4096 cores. The parameters are the following: $N_r = 128, N_\theta = 256, N_\varphi = 128, N_{v_\parallel} = 64, N_\mu = 32$. In Table 2, timings for a bigger test case are presented. Its size is $N_r = 512, N_\theta = 512, N_\varphi = 128, N_{v_\parallel} = 128, N_\mu = 32$.

For this second case, the testbed was composed of 4k to 64k cores. In Tables, the `io[1-4]` steps state for communications of task[1-4], whereas `comp[1-4]` stand for computation costs of task[1-4]. The Vlasov solver is parallelized using a domain decomposition in μ, r, θ. The number of processes #P equals $\mathrm{N}_\mu \times \mathrm{p_r} \times \mathrm{p}_\theta$.

General Observations. We map a single MPI process per node. The main idea for this OpenMP parallelization has been to target φ loops. This approach is efficient for the computation task 1 of QN solver. But in the tasks 3, 4, this strategy competes with the MPI parallelization that also parallelizes over φ. Thus, above N_φ cores, no parallelization gain is expected. This fact is not the hardest constraint up to now: communication costs are the critical overhead, much more than computation distribution for these tasks (see Table 2). A recent improvement has been to add a parallelization over μ in task 2. Even if this change adds a unecessary communication (`io2`), it is worthwhile on large platforms. Notably, we see that `comp2` scales beyond $N_\varphi = 128$ cores in Table 2.

Comments for the Small Case. In Table 1, the communication costs for exchanging $\tilde{\rho}_1$ values (`io1` - task 1) is reduced along with the involved number of nodes. This is explained by the fact that the cumulative network bandwidth increases with larger number of nodes, while the amount of data exchanged remains the same. The communication cost associated with `io3` is mainly composed of synchronization of nodes and broadcasting the 2D slice $\langle\tilde{\rho}\rangle_\theta\,(r = *, \varphi = *)$. The `io4` communication involves a selective send of Φ slices to each process; and one can note the same decreasing behaviour depending on the number of cores observed for `io1`. The `comp1` calculation is a big CPU consumer; it scales well with the number of cores. The `comp3` is negligible time and `comp4` is a small computation step, time measurements are nearly constant for all number of cores shown. As already said, N_φ cores is the upper bound of the parallel decomposition here.

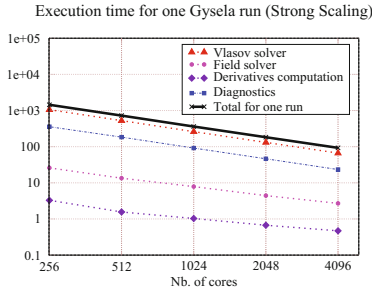


Fig. 3. Small case - GYSELA timings

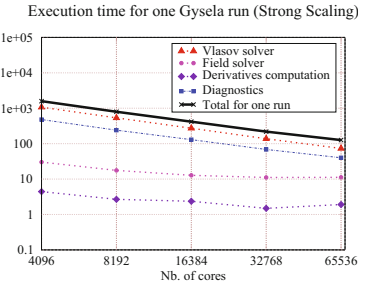


Fig. 4. Big case - GYSELA timings

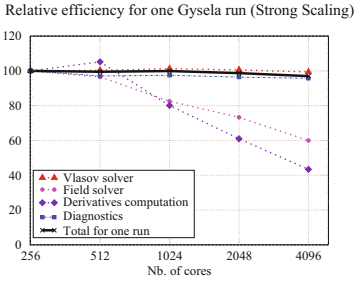


Fig. 5. Small case, GYSELA efficiency

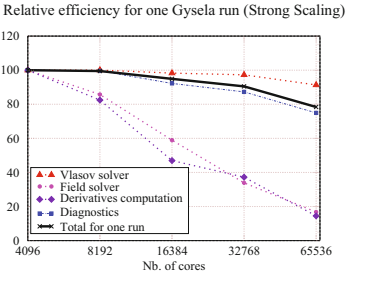


Fig. 6. Big case, GYSELA efficiency

Table 3. Memory consumption (in GB) on each node by GYSELA, big case

Nb. cores	4k	8k	16k	32k	64k
Previous version					
3D data struct.	2.79	2.75	2.73	2.72	2.72
All data struct.	11.55	8.07	6.38	5.48	5.02

Nb. cores	4k	8k	16k	32k	64k
Present version					
3D data struct.	1.07	0.82	0.58	0.39	0.24
All data struct.	9.50	5.83	3.93	2.86	2.27

The relative efficiency for the overall QN solver is 60% at 4k cores which is a good result for this solver that collects/redistributes data between all cores.

Comments for the big case. The relative speedups shown in Table 2 consider as a reference the execution times on 4k cores. Communication costs are larger than in the small test case. Badly `comp2` and `comp4` do not scale well. Only `comp1` and `io4` parts behave as we wish. In future works, we expect improving the scalability of this algorithm: reduction of communication costs is the first candidate (we investigate compression methods). Even if improvements can be found, a good property of this solver is that execution time globally decreases along with the number of cores, and does not explode (for example due to growing communication costs). We see in Fig. 6 that this cheap cost of the QN solver brings a good overall scalability. GYSELA reaches 78% of relative efficiency at 64k cores. In Fig. 3 and 4, timings for short runs of GYSELA are presented (initialization step is omitted). The field solver and derivatives computation of the

gyroaveraged Φ are low compared to Vlasov solver and diagnostics costs. Then, their limited scalability at very large number of cores, does not impact significantly the scalability of the overall code. Let us remark the excellent scalability for the small case (Fig. 5) with 97% of overall relative efficiency at 4k cores.

Memory Scalability. Thanks to the parallel algorithms described, we avoid the storage of complete 3D field data in memory, but we need only distributed 3D data. The threads inside a MPI process share these 3D data. In addition to this data distribution, another modification has occured in the present version: a set of different 3D and 2D data buffers have been removed and replaced by a unique large 1D array. The Table 3 shows the decrease in memory usage.

Conclusion

We describe the parallelization of a quasineutral Poisson solver used into a full-f gyrokinetic 5D simulator. The parallel performance of the numerical method is demonstrated. It achieves good parallel computation scalability up to 64k cores with a OpenMP/MPI approach. The coupling of the quasineutral solver and the Vlasov code is improved here by reducing communication costs and introducing better computation distribution compared to previous results [6,2,7]. The modifications result also in savings in the memory occupancy, which is a big issue when physicists wish to run large case.

Acknowledgments. This work was partially supported by the ANR GYPSI contract. The authors thanks to C.S. Chang for giving us access to Jaguar, project ref. FUS022. We wish to thank also C. Passeron for her constant help in developing and porting GYSELA code.

References

1. Görler, T., Lapillonne, X., Brunner, S., Dannert, T., Jenko, F., Merz, F., Told, D.: The global version of the gyrokinetic turbulence code gene. J. Comput. Physics 230(18), 7053–7071 (2011)
2. Grandgirard, V., Brunetti, M., Bertrand, P., Besse, N., Garbet, X., Ghendrih, P., Manfredi, G., Sarazin, Y., Sauter, O., Sonnendrucker, E., Vaclavik, J., Villard, L.: A drift-kinetic Semi-Lagrangian 4D code for ion turbulence simulation. Journal of Computational Physics 217(2), 395–423 (2006)
3. Grandgirard, V., Sarazin, Y., Garbet, X., Dif-Pradalier, G., Ghendrih, P., Crouseilles, N., Latu, G., Sonnendrucker, E., Besse, N., Bertrand, P.: Computing ITG turbulence with a full-f semi-Lagrangian code. Communications in Nonlinear Science and Numerical Simulation 13(1), 81–87 (2008)
4. Hahm, T.S.: Nonlinear gyrokinetic equations for tokamak microturbulence. Physics of Fluids 31(9), 2670–2673 (1988)
5. Jolliet, S., McMillan, B., Villard, L., Vernay, T., Angelino, P., Tran, T., Brunner, S., Bottino, A., Idomura, Y.: Parallel filtering in global gyrokinetic simulations. Journal of Computational Physics (2011) (in press)

6. Latu, G., Crouseilles, N., Grandgirard, V., Sonnendrücker, E.: Gyrokinetic Semi-lagrangian Parallel Simulation Using a Hybrid OpenMP/MPI Programming. In: Cappello, F., Herault, T., Dongarra, J. (eds.) PVM/MPI 2007. LNCS, vol. 4757, pp. 356–364. Springer, Heidelberg (2007)
7. Latu, G., Grandgirard, V., Crouseilles, N., Belaouar, R., Sonnendrucker, E.: Some parallel algorithms for the Quasineutrality solver of GYSELA. Research Report RR-7591, INRIA (April 2011), http://hal.inria.fr/inria-00583521/PDF/RR-7591.pdf
8. Lin, Z., Lee, W.W.: Method for solving the gyrokinetic Poisson equation in general geometry. Phys. Rev. E 52(5), 5646–5652 (1995)
9. Madduri, K., Im, E.J., Ibrahim, K., Williams, S., Ethier, S., Oliker, L.: Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms. Parallel Computing 37(9), 501–520 (2011)

Semantic-Based SLA Monitoring of Storage Resources

Renata Słota[1], Darin Nikolow[1], Paweł Młocek[1], and Jacek Kitowski[1,2]

[1] AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, Department of Computer Science, al. Mickiewicza 30, 30-059 Krakow, Poland
[2] AGH University of Science and Technology, ACC Cyfronet AGH, ul. Nawojki 11, 30-950 Krakow, Poland
{darin,rena,kito}@agh.edu.pl

Abstract. Storage systems play a key role in any modern data center influencing the overall performance and availability of applications and services, especially, if they deal with large volumes of data. Clients of storage services may have expectations concerning the QoS, which may be formally expressed in a contract called SLA. SLA monitoring is essential part of the automated SLA management process. In this paper we present a method of applying ontologies for SLA monitoring of storage related services.

1 Introduction

Storage systems play a key role in any modern data center influencing the overall performance and availability of applications and services, especially if they deal with large volumes of data. Using virtualization technologies data centers can make their resources and services available to clients – like in the Cloud environments [7,3]. Clients may have expectations concerning the QoS (Quality of Service), which may be formally expressed in a contract called SLA (Service Level Agreement) [12,10].

SLA monitoring is essential part of the automated SLA management process. It provides information about fulfilling SLA obligations, which can be further used by the other modules, e.g., replica manager [11] to improve the QoS or to stop accepting new SLAs when the resources get exhausted.

Because of the heterogeneity of MSS (Mass Storage Systems) it seems reasonable to apply semantic technologies for defining relations between different concepts in the context of MSS, QoS and SLA. In this paper we propose a method of applying ontologies for SLA monitoring of storage related services, developed as a part of OntoStor project [1]. This approach is targeted at modern distributed environments like Clouds and Grids [8].

An important assumption influencing the method of implementing the monitoring system and the relevant ontologies is that some of the QoS metrics may be calculated differently depending on the available attributes of the MSS being monitored. We use the term *abstract QoS metric* to describe a set of metrics

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 232–241, 2012.

sharing common meaning (measuring the same property of a resource, using the same unit) but being calculated differently, based on the availability of monitoring information for a particular resource. Another important assumption is that the method of obtaining the QoS attributes can also differ between MSS.

The rest of the paper is organized as follows. State of the art is presented in the next section. Then we define QoS metrics in section three and present the SLA monitoring system and the relevant ontologies developed within Ontostor project in section four. Example use cases are presented in section five and the last section concludes the paper.

2 State of the Art

The problems related to management of SLAs, QoS metrics and the usage of ontologies have been studied in many research works. Below we present selected papers concerning the usage of ontologies in the field of SLA management in contrast with our research. In [4] evaluation of QoS ontology research work is done and an initiative for creation of an unified QoS ontology is presented. In [5] a comprehensive ontology-based model of SLA is proposed. The model is used for automatic generation of QoS monitors. In [6] an ontology for describing SLAs is presented and discussed in the context of the telecommunications domain.

Due to the heterogeneity of MSS, which implies different methods of obtaining common QoS metrics we started a research on SLA monitoring using a previously developed common model of MSS including the relevant ontology describing performance related parameters for MSS [9], created within the OntoStor project [1].

The created ontology is modeled similarly to [4]. However, some core concepts are different, our ontology provides a way to model abstract QoS metrics, which can be measured in different ways. This allows to choose suitable method of measurement of a metric for a particular set of available monitoring information, using an ontology reasoning engine. We also introduced domain specific ontology for specifying QoS requirements for MSS.

3 Storage QoS metrics

In order to provide desirable level of QoS it is important to define how to measure the level of QoS. Several QoS metrics regarding storage systems performance are presented in this section. These QoS metrics are calculated from the monitoring parameters (further called also attributes) obtained from the low-level monitoring system and derived by sensors (see Fig.1).

The default QoS metrics for MSS defined in [1] and implemented in the presented study are presented in Table 1. The following notation is used for the metrics description: $AttributeName(t, r)$ – the monitored value of attribute $AttribiteName$ at time t for the resource r; R – set of resources for which the metric value is calculated; t_0 – current time; T – time period for which the metric value is calculated.

Table 1. QoS metrics for storage systems

QoS metric	**Metric definition**
Total available capacity	$\sum_{r \in R} TotalCapacity(t_0, r)$
Free capacity	$\sum_{r \in R} FreeCapacity(t_0, r)$
$w(t,r) \in \{0,1\} \wedge w(t,r) = 1 \Leftrightarrow IsWorking(t,r)$	
Uptime of all resources	$\underset{t_0-T<t\leq t_0}{average} \left(\min_{r \in R} (w(t,r)) \right)$
Uptime of any resource	$\underset{t_0-T<t\leq t_0}{average} \left(\max_{r \in R} (w(t,r)) \right)$
$rtr_r(t) \in \{AverageReadTransferRate(t,r), CurrentReadTransferRate(t,r),$ $max(CurrentReadTransferRate(t,r), DiskstatReadTransferRate(t,r))\}$	
Average read transfer rate	$\underset{t_0-T<t\leq t_0}{average} \left(\sum_{r \in R} rtr_r(t) \right)$
Minimal read transfer rate	$\underset{t_0-T<t\leq t_0}{min} \left(\sum_{r \in R} rtr_r(t) \right)$
$wtr_r(t) \in \{AverageWriteTransferRate(t,r), CurrentWriteTransferRate(t,r),$ $max(CurrentWriteTransferRate(t,r), DiskstatWriteTransferRate(t,r))\}$	
Average write transfer rate	$\underset{t_0-T<t\leq t_0}{average} \left(\sum_{r \in R} wtr_r(t) \right)$
Minimal write transfer rate	$\underset{t_0-T<t\leq t_0}{min} \left(\sum_{r \in R} wtr_r(t) \right)$

Average read transfer rate is an example of a metric with multiple methods of measurement. Depending on the type of a storage system, it may provide different type of information about its read transfer rate: *AverageReadTransferRate* is predetermined average transfer rate for the device, *CurrentReadTransferRate* is based on periodical active measurement and *DiskstatReadTransferRate* is additional information based on real usage of the device, which may be more accurate than the one obtained by active measurement. Depending on which of these monitoring attributes are available, the system chooses suitable method of calculating the metric.

4 Semantic-Based SLA Monitoring

The SLA monitoring system is designed to monitor fulfillment of SLA contracts. The contracts are expressed as a set of requirements – constrains on the value of specified QoS metrics for some storage resources. A QoS metric is a high-level, abstract entity and can be applicable for various different types of storage systems. This is achieved by defining a QoS metric with several different methods of calculation, each of these applicable to a different type of storage resources. The semantic way of describing methods of calculation, resources and monitoring systems allows to automatically find a method suitable for a given resource, using a semantic reasoner. The abstract way of defining QoS metrics and the semantic approach make the system easily extensible to support new metrics, types of resources or monitoring systems.

4.1 Overall System Design

The proposed system consists of 2 main parts, together with accompanying ontologies (see Fig. 1, area marked with dotted border): (1) QoS Metric Mapper, which is responsible for mapping an abstract semantic metric definition onto some concrete expression combining monitored attributes of a specified resource to calculate value of the metric; (2) SLA Monitor, which allows monitoring of SLA contracts submitted as ontological documents. The parts and the way they communicate are described in the ontology.

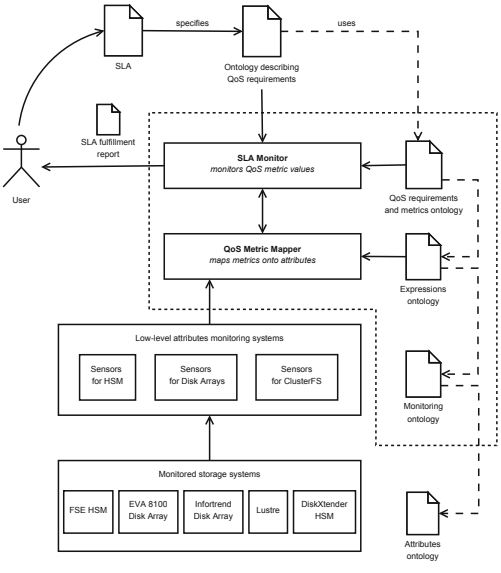


Fig. 1. Overview of the SLA monitoring system architecture

4.2 Description of System Components

The SLA Monitor analyzes the contract ontology. For each defined requirement it queries the QoS metric mapper to obtain an expression representing a metric for all specified resources. Metric mapper breaks this down to some combination of metrics, each for every single resource. Metric expression for the single resource is then mapped to monitoring systems queries.

The core of the system is the QoS Metric Mapper. Its most important feature is the ability to choose optimal way of calculating a particular metric for a particular resource, based on the available monitored attributes of the resource. Moreover, the metric may be expressed using the monitored attributes obtained from different systems monitoring the resource.

Each monitored storage system is described in an ontology specified in the contract ontology. This description provides various information on the resource, like: which monitoring systems monitor the resource, how to communicate with

this monitoring system and how to get the value of each attribute specified for the resource. This information is used by appropriate monitoring system adapter to provide monitoring information to the QoS Metric Mapper.

4.3 Ontologies

Ontologies in this system play important role, as they provide a way to describe information exchanged within the system. Due to a large number of available tools and popularity, OWL (RDF/XML) was chosen as a ontology language. The used tools include: protégé – free, Open Source ontology editor, OWLAPI – Java API and reference implementation for manipulating OWL ontologies, Pellet – semantic reasoning engine for OWL ontologies for Java.

Areas of application of the ontology in this system are as follows:

1. Description and classification of storage resources and their attributes.
2. Description of monitoring systems, representation of resources in this systems and monitored attributes of the resources.
3. Definitions of QoS metric and methods of calculating them based on resources' attributes.
4. Way of describing QoS requirements in contracts.

Following is a description of the created OntoStor ontologies (see also Fig.2) that satisfy these requirements.

Attributes Ontology (OntoStor-ATN). This ontology provides a model of storage resources and their attributes. It allows to describe storage systems of different types (HSM (Hierarchical Storage Management)), disk arrays, local drives) and abstract attributes associated with these systems (see model in [9]).

Monitoring and Expressions Ontology (OntoStor-monitoring). It contains few types of concepts: MeasurementConcepts, MonitoringConcepts and ExpressionConcepts (see Fig. 2).

MeasurementConcepts describe a system of measurements, i.e. simple and complex units, dimensions of the units, and conversion rates between the units. This allows performing calculations using different sources with different units.

MonitoringConcepts describe monitoring systems and their types, monitored storage resources and monitored attributes. Other ontologies deriving from this concepts describe particular monitoring system types and any information specific to them. Each monitoring system can provide a subset of attributes described in OntoStor-ATN for a particular monitored storage system.

ExpressionConcepts provide means of describing arithmetic, Boolean or even more generic expressions. Expressions of these types are used to construct definitions of QoS metrics and methods of calculating them.

QoS Requirements Ontology (OntoStor-qos). This ontology provides concepts which allow to describe SLA contracts and QoS requirements within these contracts, QoS metrics and methods of calculating them. The ontology also defines some higher-level expressions, like a QoS metric value for single resource or a QoS metric value for multiple resources.

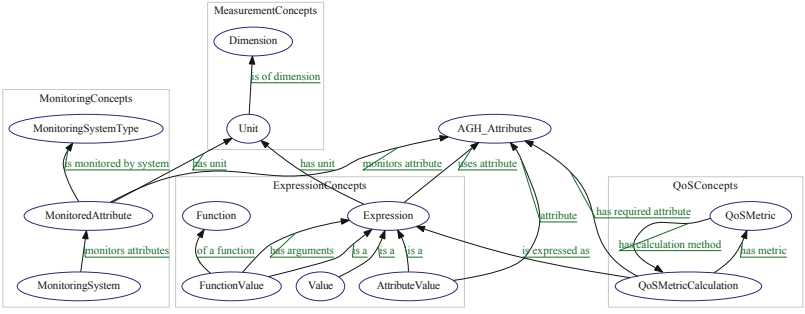


Fig. 2. Main concepts in OntoStor ontologies

Metrics Ontology (OntoStor-metrics). This ontology contains the definition of the QoS metrics described in section 3, using ExpressionConcepts from OntoStor-monitoring ontology and metrics-related concepts from OntoStor-qos.

4.4 Example – Analyzing and Monitoring of a Requirement

The following ontology example describes a single QoS requirement from an example SLA contract ontology. The requirement is met when the value of `averageReadTransferRate` metric for resources `agh_da0` and `agh_hsm0` is greater than 300 MB/s. The example QoS requirement:

```
<owl:NamedIndividual rdf:about="&sla;averageReadTransferRate">
    <rdf:type rdf:resource="&qos;QoSRequirement"/>
    <qos:isValidatedUsing rdf:resource="&sla;bwGt300MBps"/>
    <qos:hasMetricValue rdf:resource="&sla;testMetricValue"/>
</owl:NamedIndividual>
```

defines this constrain:

```
<owl:NamedIndividual rdf:about="&sla;bwGt300MBps">
    <rdf:type rdf:resource="&monitoring;NamedParametersFunctionValue"/>
    <monitoring:hasArguments rdf:resource="&sla;requirementMetricValue"/>
    <monitoring:hasArguments rdf:resource="&sla;value300MBps"/>
    <monitoring:ofFunction rdf:resource="&monitoring;greaterThan"/>
</owl:NamedIndividual>
```

on this QoS metric value:

```
<owl:NamedIndividual rdf:about="&sla;testMetricValue">
    <rdf:type rdf:resource="&qos;QoSMetricValue"/>
    <qos:forResources rdf:resource="&sla;agh_da0"/>
    <qos:forResources rdf:resource="&sla;agh_hsm0"/>
    <qos:ofMetric rdf:resource="&metrics;averageReadTransferRate"/>
</owl:NamedIndividual>
```

Analyzing. The purpose of analysis of this requirement is to create its calculable representation. First, metric value expression (`&sla;testMetricValue`) is analyzed. It is done in the following steps:

1. For each specified resource (in this case, `&sla;agh_da0` and `&sla;agh_hsm0`) the QoS Metric Mapper queries the ontology reasoner to get possible metric's methods of calculation for the resource (based on availability of monitored attributes). This is done using following query:

```
QoSMetricCalculation
that hasMetric value <metric>
and calculationRequires
    only (Attribute that isProvidedByResource some
            (MonitoredResource that hasResource value <resource>))
                and inverse monitoringSystemProvides some
                    (MonitoringSystem that inverse isMonitoredBySystem some
                        (MonitoredResource that hasResource value <resource>)))
```

2. Ontology reasoner returns all available methods of calculating metric value for specified resource, based on available monitored resource's attributes. In our case there are three possible methods: *AverageReadTransferRate*, *CurrentReadTransferRate* or *DiskstatReadTransferRate* (see Table 1).
3. QoS Metric Mapper chooses one way of measuring the metric.
4. QoS Metric Mapper queries the ontology to retrieve a description of the expression calculating the metric value.
5. Mapper creates Java objects corresponding to the elements of the expression. This may be function invocations, constant values or monitored attribute values, arranged together as an expression tree. In order to create objects representing monitored attribute value, the system queries the ontology for a way of monitoring the attributes required to calculate this metric and initializes appropriate monitoring systems adapters (see Fig. 3).
6. Now, each evaluation of the created expression returns the QoS metric value for the resources.
7. Expression used to validate the requirement is created, in similar way as in point 5. In this example it is simple comparison, which yields positive result when QoS metric value is greater than specified constant value (in this case, 300MB/s).
8. The result of the analysis of a requirement is a calculable expression representing the QoS metric value for the specified resources and an expression which given the QoS metric value, validates whether the requirement is met.

Monitoring. Once the expression representing the QoS metric value and the expression validating the QoS requirement are created by QoS Metric Mapper, the SLA monitoring is simple:

1. SLA monitor queries monitoring systems (through adapters) to retrieve value of attributes required to calculate the QoS metric value for each resource.
2. This values are passed to the already created expressions calculating metric value using predetermined method. Evaluation of this expression yields the value of the QoS metric.
3. SLA monitor passes the calculated QoS metric value to the validating expression. In this case, simple comparison is performed. Positive result means that the requirement is fulfilled.
4. SLA monitor stores the QoS metric value and the result of validation to present them to the user in a useful way.

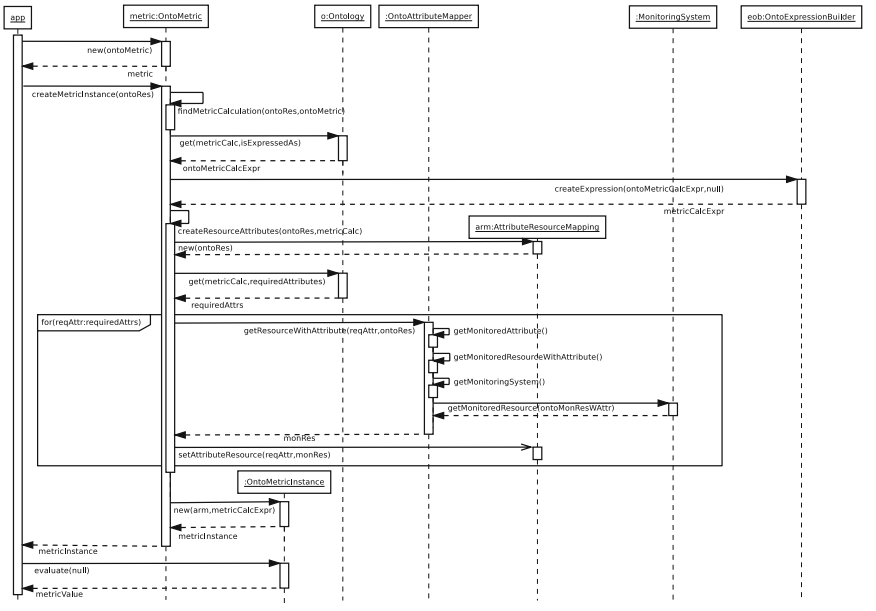


Fig. 3. Sequence diagram of metric instantiating

5 Use Cases

Typical use case of this system may be described in following steps:

1. Service provider creates an ontology describing their storage systems (using OntoStor-ATN ontology). This may involve extending OntoStor-ATN ontology to include custom attributes.
2. Service provider creates an ontology describing how these storage systems are monitored. If used monitoring technology is not already supported by the system, service provider needs to extend OntoStor-monitoring ontology and provide Java plugin to interface with this monitoring system.
3. Service provider reviews available metrics and may optionally add new ones. This involves only extension of the base OntoStor-metrics ontology, since methods of measuring metrics are fully specified within ontology.
4. Customer requests a service with particular QoS, and together with service provider they specify SLA contract ontology. It includes a set of requirements, being a constrains on the values of QoS metrics for resources.
5. The contract is submitted to the SLA monitoring system, and is being continuously monitored. QoS metrics' method of calculation is automatically matched to particular resources, based on the definition of the method of calculation and available monitoring information for the resource.
6. Results of the monitoring are presented. Users can view percentage of time when each requirement was satisfied, together with past and present values of the QoS metrics. Overall contract satisfaction is currently calculated as a percentage of time when all the requirements were satisfied.

5.1 Deployment Example

The SLA monitoring system was deployed together with a simple web interface. Few example contracts were tested, regarding different types of storage systems: local disks, HSM system and disk arrays. Monitored attributes of this resources were available through two different monitoring systems: Gemini2 [2] and HSM systems monitor with a RESTful webservice interface.

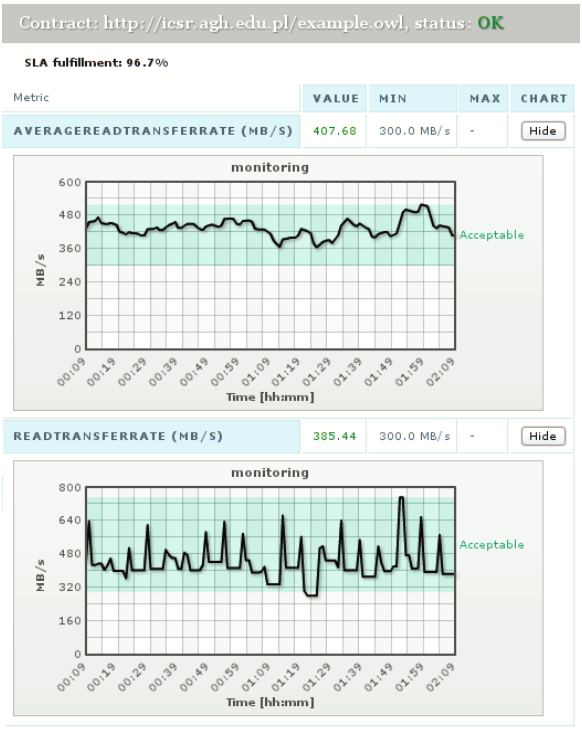


Fig. 4. SLA fulfillment monitoring - test results

In the screenshot (Fig. 4) a result of monitoring an example SLA contract is shown. Excerpts from this contract (defining of a requirement) are shown in section 4.4. The contract consists of 2 requirements, constraining 2 metric values. First metric value, *read transfer rate* is *averageReadTransferRate* without averaging over time. Second metric value, *average read transfer rate* is the same as previous, but averaged over 10 minutes, both calculated for two different resources: a disk array and a HSM system. Each resource is also monitored using different monitoring system. Values of the metrics is required by the contract to be over 300MB/s. Both values are higher, therefore the contract is satisfied. The contract was satisfied 96.7% of the time.

6 Conclusion

In this paper we have presented an approach for semantic-based SLA monitoring of storage resources. The main contribution of this research is proposing a method of specifying abstract QoS metrics using ontologies. The semantic way of describing methods of calculation, storage resources and monitoring systems allows to automatically obtain a method of QoS metric calculation suitable for the given storage resource. The presented approach allows to use one QoS metric set for different storage resources due to possibility to fit the way of metric calculation according to the available storage resource monitoring attributes.

Acknowledgments. This research is partially supported by the MNiSW grant nr N N516 405535 and AGH-UST grant nr 11.11.120.865.

References

1. OntoStor project (May 17, 2011), http://www.icsr.agh.edu.pl/ontostor/
2. Balis, B., Kowalewski, B., Bubak, M.: Real-time Grid monitoring based on complex event processing. Future Generation Computer Systems 27, 1103–1112 (2011)
3. Buyya, R., Yeo, C., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25, 599–616 (2009)
4. Dobson, G., Sanchez, A.: Towards Unified QoS/SLA Ontologies. In: Proc. of the IEEE Serv. Comp. Workshops, pp. 169–174. IEEE Computer Society (2006)
5. Fakhfakh, K., et al.: A Comprehensive Ontology-Based Approach for SLA Obligations Monitoring. In: Proc. of the 2nd Int. Conf. on Advanced Engineering Computing and Applications in Sciences, pp. 217–222. IEEE Computer Society (2008)
6. Green, L.: Service level agreements: an ontological approach. In: Proc. of the 8th Int. Conf. on Electronic Commerce, ICEC 2006. pp. 185–194. ACM, New York (2006)
7. Kant, K.: Data center evolution. Comput. Netw. 53, 2939–2965 (2009)
8. Marco, J., Campos, I., Coterillo, I., et al.: The interactive European grid: Project objectives and achievements. Computing and Informatics 27, 161–171 (2008)
9. Polak, S., Nikolow, D., Słota, R., Kitowski, J.: Modeling Storage System Performance for Data Management in Cloud Environment using Ontology. In: Proc. of IWSI 2011 Int. Workshop on Semantic Interoperability, pp. 54–63. SciTePress, Rome (2011)
10. Skitał, Ł., Janusz, M., Słota, R., Kitowski, J.: Service Level Agreement Metrics for Real-Time Application on the Grid. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 798–806. Springer, Heidelberg (2008)
11. Słota, R., Nikolow, D., Skitał, Ł., Kitowski, J.: Implementation of Replication Methods in the Grid Environment. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 474–484. Springer, Heidelberg (2005)
12. Stelmach, M., Kryza, B., Słota, R., Kitowski, J.: Distributed contract negotiation system for virtual organizations. Procedia Computer Science 4, 2206–2215 (2011); Proc. of the Int. Conf. on Computational Science, ICCS 2011

The Generalization of AQM Algorithms for Queueing Systems with Bounded Capacity

Oleg Tikhonenko[1] and Wojciech M. Kempa[2]

[1] Institute of Mathematics, Czestochowa University of Technology, Poland
oleg.tikhonenko@im.pcz.pl
[2] Institute of Mathematics, Silesian University of Technology, Poland
wojciech.kempa@polsl.pl

Abstract. A queueing system of the $M/M/1/(\infty, V)$ type with generally distributed packet volumes and bounded capacity (total packets volume) is considered. The queue length is controlled by means of the accepting function that enqueues the arriving packet with probability depending on the free capacity volume in the system at the pre-arrival epoch. Explicit representations for stationary probabilities are derived via solving the system of differential equations. Sample numerical results are attached in which stationary queue-size distributions with and without dropping packets are compared.

Keywords: AQM algorithms, Loss Probability, Numerical Laplace transform inversion, Queue-size distribution.

1 Introduction

Queueing systems with finite buffer capacities have numerous applications in analysis of modern telecommunication and computer networks. In particular, they are used for modelling processes occurring in nodes of networks, like Internet routers or ATM switches. One of the major practical problems occurring in such networks is the phenomenon of buffer overflow, which results in temporary blocking of data transmission, such as e.g. IP packets (it corresponds to stopping the service process in the appropriate queueing system). The prevention of this problem by increasing the capacity of the buffer can result in a significant prolongation of the real time of transmission (the sojourn time of the packet in the corresponding queueing system increases due to increasing the maximal queue length).

One of the tools for avoiding the risk of buffer saturation is using the Active Queue Management (AQM) algorithms. In [8] one of such scheme called Random Early Detection (RED) was proposed. The idea of RED procedure is in introducing a mechanism for monitoring the current queue size by a drop function that can depend on the instantaneous queue size n. This function, with some positive probability $d(n)$, rejects the incoming packets even when the buffer is not saturated. A typical drop function (see [4]) is linearly increasing from 0 to

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 242–251, 2012.

some value $d_{\max}$, for queue sizes between $q_{\min}$ and $q_{\max}$. Below $q_{\min}$ the enqueueing process is not controlled, and above $q_{\max}$ the incoming packets are dropped with probability 1. Different modifications of the classical RED algorithm can be found in the literature. In [10] the linear drop function is replaced with an exponential and in [14] – by a quadratic one. A dynamic RED scheme (DRED) is proposed in [3] and REM algorithm is introduced in [2]. In [7] a BLUE algorithm is proposed. In this scheme the incoming packets are dropping with certain constant initial probability that is increased by ϵ when the buffer overflows and decreased by ϵ when the queue empties.

One of the fundamental characteristics of each queueing system is the queue-size distribution. In the paper we study the stationary queue-size distribution in the system in that the incoming packets can have different volumes which are generally distributed random variables, at the same time the total packets volume in the system is bounded. Instead of the classical drop function we introduce the accepting function that enqueues the incoming packet with probability that depends on the free capacity volume in the system at the pre-arrivals epoch.

Such a system is a natural generalization of the AQM-modelled system with batch arrivals to the case in that the volume of the packet can be continuously distributed. Evidently, due to this extension the model becomes more applicable in the performance analysis e.g. of telecommunication and computer networks. The analysis generalizes partially results obtained in [9] where the classical $M^X/M/1/N$ finite-buffer queue with batch arrivals and AQM schemes was investigated. The generalization is based on the theory of queueing systems with random volume customers (see e.g. [11,12]).

The paper is organized as follows. In Section 2 we give a precise description of the system and introduce necessary notation. In Section 3 we introduce the Markovian process describing the evolution of the system. In Section 4 we write down the system of differential equations for stationary queue-size probabilities and find its solution. As a special case formulae for stationary queue-size distribution and loss probability are given for the $M/M/1/(\infty, V)$ system without dropping packets. Section 5 is devoted to numerical computations: we compare stationary probabilities in sample systems with and without AQM algorithm. In this section we use one of algorithms for numerical Laplace transform inversion. The last Section 6 contains conclusions.

2 System Description and Necessary Notation

Let us consider the system of $M/M/1/(\infty, V)$ type [13]. Denote by a the arrival rate of incoming packets. Each packet is characterized by its random volume ζ that is generally distributed with the distribution function (d.f.) $L(x) = = \mathbf{P}\{\zeta < x\}$. The service time ξ of the packet is independent of its volume and exponentially distributed with parameter μ, so we denote $B(t) = \mathbf{P}\{\xi < t\} = = 1-e^{-\mu t}$. Denote by $\sigma(t)$ the total volume of all packets present in the system at time instant t. It is assumed that the total volume is bounded by a non-random positive value (capacity volume) V.

Denote by τ any (fixed) arrival epoch in the stationary state of the system. The packet arriving at τ is lost if its volume x satisfies condition $x > V - y$, where $y = \sigma(\tau^-)$. Introduce a right-hand continuous and nondecreasing function $r(v)$, defined on the interval $[0, V]$ such that $r(V) \leq 1$, $r(0) \geq 0$. The function $r(v)$ represents the probability that the incoming packet is enqueued in the system if the free volume of the system at the pre-arrival epoch equals v units. In other words, if the packet with volume x arrives at time τ and $\sigma(\tau^-) = y$, then it will be accepted for service with probability $r(V - y)$ and dropped with probability $1 - r(V - y)$. Moreover, the incoming packet will be lost if $x > V - y$. Let us note that, if $d(y)$ is the value of the classical drop function at the pre-arrival epoch, then we have $r(V - y) = 1 - d(y)$.

Let $\eta(t)$ be the number of packets present in the system at time t. If the packet arriving at time τ is lost (dropped) then we have $\eta(\tau) = \eta(\tau^-)$ and $\sigma(\tau) = \sigma(\tau^-)$. Otherwise we have $\eta(\tau) = \eta(\tau^-) + 1$ and $\sigma(\tau) = \sigma(\tau^-) + x$.

3 Stochastic Process Describing the Evolution of the System

The evolution of the investigated queueing system can be described by means of the following Markovian process:

$$\big(\eta(t),\, \zeta_i(t),\, i = 1, ..., \eta(t)\big), \tag{1}$$

where $\zeta_i(t)$ denotes the volume of the ith packet present in the system at time t (we assume that the arriving packets are numbered according to their appearance). Components $\zeta_i(t)$ vanish if $\eta(t) = 0$. In this case, obviously, we have $\sigma(t) = 0$ and otherwise $\sigma(t) = \sum_{i=1}^{\eta(t)} \zeta_i(t)$.

The process (1) will be characterized by functions having the following probability meanings:

$$P_0(t) = \mathbf{P}\{\eta(t) = 0\}; \tag{2}$$

$$G_k(y,t)dy = \mathbf{P}\{\eta(t) = k,\, \sigma(t) \in [y, y + dy)\}, \quad k = 1, 2, ...; \tag{3}$$

$$P_k(t) = \mathbf{P}\{\eta(t) = k\} = \int_0^V G_k(y,t)dy, \quad k = 1, 2, ... \tag{4}$$

Since for any finite a, V and nonzero μ there exists the stationary state of the system, then the following limits also exist:

$$p_0 = \lim_{t\to\infty} P_0(t) = \mathbf{P}\{\eta = 0\}; \tag{5}$$

$$g_k(y)dy = \lim_{t\to\infty} G_k(y,t)dy = \mathbf{P}\{\eta = k,\, \sigma \in [y, y + dy)\}, \quad k = 1, 2, ...; \tag{6}$$

$$p_k = \lim_{t\to\infty} P_k(t) = \mathbf{P}\{\eta = k\} = \int_0^V g_k(y)dy, \quad k = 1, 2, ..., \tag{7}$$

where η and σ denote the stationary number of packets and the stationary total volume in the system respectively.

4 System of Equations for Stationary Probabilities

It is easy to show that functions (2)–(4) satisfy the following differential equations:

$$P_0'(t) = -aP_0(t)r(V)L(V) + \mu P_1(t); \tag{8}$$

$$\begin{aligned} P_k'(t) =& a\int_0^V G_{k-1}(x,t)r(V-x)L(V-x)dx \\ &- a\int_0^V G_k(x,t)r(V-x)L(V-x)dx - \mu P_k(t) + \mu P_{k+1}(t), \\ & k = 1, 2, ... \end{aligned} \tag{9}$$

Taking in (8), (9) limits as $t \to \infty$ we obtain the following system of equations for stationary probabilities p_k :

$$0 = -ap_0 r(V)L(V) + \mu p_1; \tag{10}$$

$$\begin{aligned} 0 =& a\int_0^V g_{k-1}(x)r(V-x)L(V-x)dx \\ &- a\int_0^V g_k(x)r(V-x)L(V-x)dx - \mu p_k + \mu p_{k+1}, \\ & k = 1, 2, ... \end{aligned} \tag{11}$$

Denoting by $\rho = \frac{a}{\mu}$ the traffic load of the system, from the equations (10), (11) the following representation for $g_k(y)dy$ can be written (that can be formally proved by direct calculations):

$$g_k(y)dy = \rho^k p_0 d\big[(r \cdot L)^{k*}(y)\big], \quad k = 1, 2, ..., \tag{12}$$

where the notation $H^{k*}(x)$ denotes the k-fold Stieltjes convolution of the function $H(x)$ with itself i.e.

$$H^{0*}(x) = 1, \qquad H^{k*}(x) = \int_0^x H^{(k-1)*}(x-y)dH(y), \quad k = 1, 2, ... \tag{13}$$

From (12) we get

$$p_k = p_0 \rho^k (r \cdot L)^{k*}(V), \quad k = 1, 2, ... \tag{14}$$

The value of p_0 can be found from the normalization identity:

$$p_0 = \Big(\sum_{k=0}^{\infty} \rho^k (r \cdot L)^{k*}(V)\Big)^{-1}. \tag{15}$$

The loss probability P can be obtained from the equilibrium equation, thus it equals

$$P = 1 - \frac{1 - p_0}{\rho}. \tag{16}$$

Assume now that the arriving packet is being accepted (or dropped) in dependence only on the number of packets present in the system just before its arrival and, besides, the total number of packets in the system is bounded by $m+1$ (m places in the buffer queue and one place for service). In fact, we obtain the system of the $M/M/1/m+1$ type in that the incoming packet is being qualified for service with probability $r_k = r(m+1-k)$, where k denotes the number of packets at the pre-arrival epoch.

Such a system is a special type of that one considered earlier, under the assumption that all packets have the same volume equal to one ($\zeta \equiv 1$) and $V = m+1$.

It is easy to show that the stationary probabilities in the simplified system can be found as

$$p_k = \frac{\rho^k \prod_{i=0}^{k-1} r_i}{1 + \sum_{j=1}^{m+1} \rho^j \prod_{i=0}^{j-1} r_i}, \quad k = 0, ..., m+1. \tag{17}$$

Note that the last result corresponds to that obtained in [9] by taking $r_k = = 1 - d(k)$, where $d(k)$ denotes the value of the dropping function for k packets present in the system.

Finally, let us note that taking $r(v) = 1$, $v \in [0, V]$, we obtain a "pure" system of the $M/M/1/(\infty, V)$ type (without dropping packets). In such a system stationary probabilities $\widehat{p}_k$ can be written as follows (see e.g. [13]):

$$\widehat{p}_k = \widehat{p}_0 \rho^k L^{k*}(V), \quad k = 1, 2, ... \tag{18}$$

and

$$\widehat{p}_0 = \Big(\sum_{k=0}^{\infty} \rho^k L^{k*}(V) \Big)^{-1}. \tag{19}$$

Similarly, the loss probability $\widehat{P}$ in the $M/M/1/(\infty, V)$ system without AQM algorithm can be written as (compare (16))

$$P = 1 - \frac{1 - \widehat{p}_0}{\rho}. \tag{20}$$

In particular, if the volume of an arriving packet is exponentially distributed with parameter f i.e. $L(x) = 1 - e^{-fx}$, $x > 0$, then probabilities (18)–(20) take the following forms respectively (see [13]):

$$\widehat{p}_k = \widehat{p}_0 \rho^k \Big[1 - e^{-fV} \sum_{j=0}^{k-1} \frac{(fV)^j}{j!} \Big], \tag{21}$$

$$\widehat{p}_0 = \begin{cases} \frac{1-\rho}{1-\rho e^{-(1-\rho)fV}}, & \text{for } \rho \neq 1, \\ (1+fV)^{-1}, & \text{for } \rho = 1. \end{cases} \tag{22}$$

The loss probability $\widehat{P}$ equals

$$\widehat{P} = \begin{cases} \frac{1-\rho}{e^{(1-\rho)fV} - \rho}, & \text{for } \rho \neq 1, \\ (1+fV)^{-1}, & \text{for } \rho = 1. \end{cases} \tag{23}$$

5 Numerical Computations

The main goal of this section is to compare stationary probabilities p_k and $\widehat{p}_k$ in sample systems with and without dropping packets. In numerical computations we will use the algorithm of numerical Laplace transform inversion that was introduced in [1]. It is based on the Bromwich inversion integral that evaluates the value of the function $F(t)$ from its Laplace transform $f(s)$ as

$$F(t) = \frac{1}{2\pi i} \int_{b-i\infty}^{b+i\infty} e^{st} f(s) ds, \tag{24}$$

and Euler summation formula (see [1] and [5] for more details).

In numerical computations we use the *Mathematica* environment. Below we present results for stationary probabilities (14), (15) and the loss probability defined in (16) for three different sample input "conditions" of the system.

(1) Let us firstly consider the system with $\rho = 2.0$ (strongly overloaded) and $V = 10$, in that the volume of the arriving packet is exponentially distributed with mean $\alpha^{-1} = 0.5$. Define the accepting function $r(v)$ in the following way:

$$r(v) = \frac{v}{V}, \quad 0 \le v \le V. \tag{25}$$

In Table 1 we present numerical results for stationary queue-size distributions in such a system considered separately with and without AQM algorithm (with accuracy of six significant digits). Moreover, stationary probabilities for the system without dropping packets are also presented in Figure 1.

Appropriate loss probabilities equal $P = 0.569850$ and $\widehat{P} = 0.500000$.

Table 1. Comparing stationary probabilities p_k and $\widehat{p}_k$ for case **(1)**

Queue size k	Stationary probability p_k	Stationary probability $\widehat{p}_k$
0	0.139700	1.03058×10^{-9}
1	0.278773	2.06116×10^{-9}
2	0.276607	4.12232×10^{-9}
3	0.178722	8.24464×10^{-9}
4	0.0841088	1.64892×10^{-8}
5	0.0305664	3.2978×10^{-8}
6	0.00889457	6.59524×10^{-8}
7	0.0021244	1.31881×10^{-7}
8	0.000424114	2.63623×10^{-7}
9	0.0000717861	5.26556×10^{-7}
10	0.0000104248	1.05004×10^{-6}

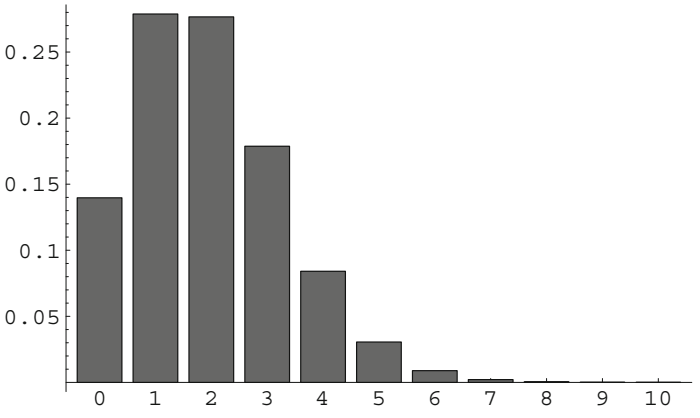


Fig. 1. Stationary probabilities p_k for case **(1)**

(2) Now let us take $L(x) = 1 - e^{-x}(1 + x)$, $x > 0$ (2-Erlang distribution with parameter 1), $V = 10$ and $\rho = 0.7$ (underloaded system). Besides, define the acceptance function $r(v)$ as follows:

$$r(v) = 1 - \frac{(v - V)^2}{V^2} \quad 0 \le v \le V. \tag{26}$$

Stationary queue-size distributions for the case of system with AQM and a "pure" one are shown in Table 2 and in Figure 2 (in darker colour for the system with dropping). Besides, the loss probabilities are $P = 0.2112227$ and $\widehat{P} = 0.0756514$ respectively.

Table 2. Comparing stationary probabilities p_k and $\widehat{p}_k$ for case **(2)**

Queue size k	Stationary probability p_k	Stationary probability $\widehat{p}_k$
0	0.448559	0.352956
1	0.313832	0.246943
2	0.171089	0.171161
3	0.0551387	0.112942
4	0.0101847	0.0660822
5	0.00111604	0.0321563
6	7.62985×10^{-5}	1.25914×10^{-2}
7	3.42022×10^{-6}	3.93966×10^{-3}
8	1.05107×10^{-7}	9.91732×10^{-4}
9	2.31908×10^{-9}	2.03357×10^{-4}
10	4.36933×10^{-11}	3.44403×10^{-5}

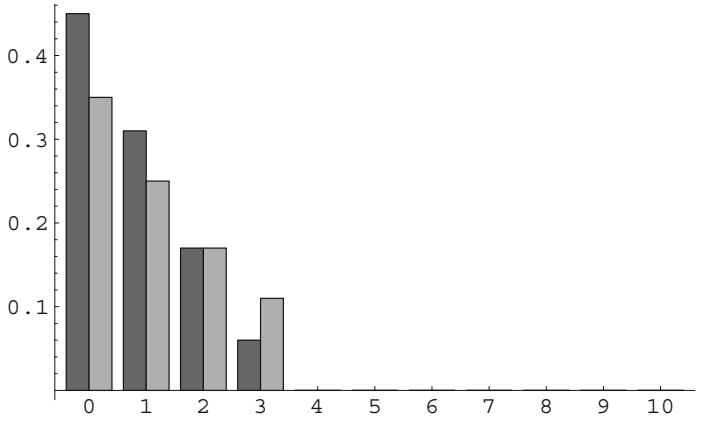


Fig. 2. Comparing stationary probabilities p_k and $\widehat{p}_k$ for case **(2)**

(3) Finally, let us consider the system in that $\rho = 0.9$ (underloaded in heavy traffic), $V = 10$ and the packet volumes have 3-Erlang distribution with parameter $\lambda = 0.5$, so we have

$$L(x) = 1 - \frac{1}{2}e^{-\frac{1}{2}x}\left(\frac{1}{4}x^2 + x + 2\right), \quad x > 0. \tag{27}$$

Choose the accepting function $r(v)$ as in the case **(2)**. In Table 3 (see also Figure 3) we present results for probabilities p_k and $\widehat{p}_k$, for $k = 0, 1, ..., 10$. Moreover, loss probabilities in the system with and without packet dropping equals $P = 0.443716$ and $\widehat{P} = 0.405351$ respectively.

Table 3. Comparing stationary probabilities p_k and $\widehat{p}_k$ for case **(3)**

Queue size k	Stationary probability p_k	Stationary probability $\widehat{p}_k$
0	0.499344	0.464816
1	0.392966	0.365793
2	0.101605	0.144591
3	0.00597565	0.0230736
4	$\times 1.08817 \times 10^{-4}$	0.00166301
5	7.83935×10^{-7}	6.21003×10^{-5}
6	2.77802×10^{-9}	1.33826×10^{-6}
7	2.40534×10^{-11}	1.81255×10^{-8}
8	1.60297×10^{-12}	1.82717×10^{-10}
9	7.88015×10^{-14}	4.30895×10^{-12}
10	2.51900×10^{-15}	3.82569×10^{-13}

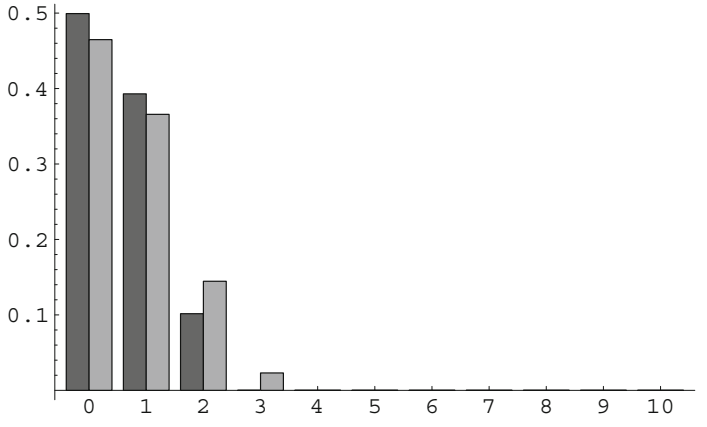


Fig. 3. Comparing stationary probabilities p_k and $\widehat{p}_k$ for case **(3)**

6 Conclusions

In the paper stationary probabilities for the $M/M/1/(\infty, V)$ queueing model with bounded total volume and the queue size controlled by a mechanism of dropping packets (AQM) are derived. For sample systems stationary distributions are compared numerically for the evolution with and without AQM algorithm. In computations an algorithm of numerical Laplace transform inversion based on Bromwich integral and Euler summation formula is used.

As numerical examples show, the introduction of a function that accepts incoming packets in dependence on the free volume of the system at pre-arrival epoch allows to reduce probability of high queue-size significantly. Thus, a suitable accepting function $r(\cdot)$ can be helpful in avoiding the buffer congestion and blocking the service of packets. However, as one can note, the loss probability in the "pure" system is lower than in the case of AQM algorithm.

References

1. Abate, J., Choudhury, G.L., Whitt, W.: An introduction to numerical transform inversion and its application to probability models. In: Grassmann, W. (ed.) Computational Probability, pp. 257–323. Kluwer, Boston (2000)
2. Athuraliya, S., Li, V.H., Low, S.H., Yin, Q.: REM: Active Queue Management. IEEE Network 15(3), 48–53 (2001)
3. Aweya, J., Ouellette, M., Montuno, D.Y., Chapman, A.: A Control theoretic approach to Active Queue Management. Comput. Netw. 36 (2001)
4. Bonald, T., May, M., Bolot, J.-C.: Analytic evaluation of RED performance. In: Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1415–1424 (2000)
5. Chydzinski, A.: Queueing characteristics for Markovian traffic models in packet-oriented networks. Silesian University of Technology Press, Gliwice (2007) (in Polish)

6. Cohen, J.W.: The single server queue. North-Holand Publishing Company, Amsterdam (1982)
7. Feng, W., Kandlur, D., Saha, D., Shin, K.: Blue: A new class of Active Queue Management algorithms. U. Michigan CSE-TR-387-99 (1999)
8. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. IEEE/ACM Transactions of Networking 1(4), 397–412 (1993)
9. Kempa, W.M.: On main characteristics of the M/M/1/N queue with single and batch arrivals and the queue size controlled by AQM algorithms. Kybernetika (to appear)
10. Liu, S., Basar, T., Srikant, R.: Exponential RED: A stabilizing AQM scheme for low- and ligh-speed TCP protocols. IEEE/ACM ToN (2005)
11. Tikhonenko, O.M.: Generalized Erlang Problem for Service Systems with Finite Total Capacity. Problems of Information Transmission 41(3), 243–253 (2005)
12. Tikhonenko, O.M.: Queueing systems of a random length demands with restrictions. Automation and Remote Control 52(10, pt. 2), 1431–1437 (1991)
13. Tikhonenko, O.: Probability Analysis of Information Systems. EXIT, Warsaw (2006) (in Polish)
14. Zhou, K., Yeung, K.L., Li, V.O.K.: Nonlinear RED: A simple yet efficient active queue management scheme. Comput. Netw. 50(18), 3784–3794 (2006)

Parallel Implementation and Scalability of Cloud Resolving EULAG Model

Andrzej A. Wyszogrodzki, Zbigniew P. Piotrowski, and Wojciech W. Grabowski

National Center for Atmospheric Research, Boulder, Co, USA
{andii,piotrows,grabow}@ucar.edu

Abstract. Progress in the development of the PetaScale implementation of the anelastic EULAG model combined with the warm-rain bulk and bin microphysics schemes, as well its application to multiscale cloud modeling, are presented. A new three-dimensional (3D) model domain decomposition is implemented to increase model performance and scalability. We investigate performance of the code on IBM BlueGene/L and Cray XT4/XE6 architectures. The scalability results show significant improvement of the new domain decomposition over the previous 2D decomposition used as the standard in many geophysical fluid flow models.

Keywords: Cloud resolving models, parallel performance, large eddy simulations.

1 Introduction

Cloud dynamics and microphysics play an essential role in the weather and climate at local and global scales. Strong inhomogeneities within clouds and critical couplings between multiphase microphysical processes and processes at larger scales (such as radiative transfer, cloud dynamics, stratified and rotating flow dynamics) involve a wide range of spatial and temporal scales. Cloud microphysics concerns latent heating that drives convective motions, with cloud particles changing their size due to condensation and evaporation. At the cloud microscale (spatial scales between a millimeter and a centimeter), individual cloud droplets are suspended in the turbulent moist air. Their sedimentation and collisions lead to the formation of drizzle and rain drops. At larger scales, energy-containing fluid-flow eddies (from tens of meters to kilometers) result in a broad range of cloud structures. At the mesoscale, Earth rotation plays the key rome in the development of mesoscale convective systems and tropical cyclones. Weather patterns associated with extratropical cyclones and anticyclones dominate synoptic variability. Monsoon circulations, large-scale convective coherences (e.g., intraseasonal oscillations such as the Madden-Julian oscillation, MJO) and the El Niño-Southern Oscillation (ENSO) are examples of planetary-scale phenomena where moist processes are critical.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 252–261, 2012.

Cloud resolving models (CRM) allow quantitative description of multiscale interactions between small-scale moist processes and larger-scale dynamics, and reduce of the number of parameterizations and assumptions. However, the computational effort is challenging, and it does not allow for covering the whole range of scales within a single model. Scales below O(10 cm) are typically investigated applying direct numerical simulation (DNS) models [1]. Scales between O(10 m) and O(10 km) are investigated by large-eddy simulation (LES) models which explicitly resolve large-scale turbulent eddies in the boundary layer while parameterizing the effect of smaller unresolved and less energetic perturbations [2]. For scales above O(100 km) up to the global scale, a range of cloud-resolving models can be used, such as, for instance, the Cloud Resolving Convection Parameterization (CRCP, [3]) or Diabatic Acceleration and REscaling (DARE, [4]).

As the resolution of CRM improves, application of PetaScale computing allows reducing the gap between different scale ranges. This requires that CRM works efficiently on contemporary and future architectures, and features high level of scalability to accommodate large number of processing cores, from $O(10^4)$ to $O(10^6)$. Here we present parallel implementation of the geophysical fluid flow model EULAG [5], in the contest of the cloud-resolving simulations at different scales. We compare scalability of a two-dimensional (2D) domain decomposition with a newly developed three-dimensional (3D) approach [6]. The physical benchmarks include the precipitating thermal and shallow cumulus field simulations based on BOMEX case studies [7].

The paper is organized as follows. Section 2 describes model formulation, and microphysics parameterizations. Section 3 describes model parallel implementation. Section 4 presents configuration of benchmark experiments. Section 5 demonstrate scalability results, while conclusions are presented in Section 6.

2 Cloud Resolving Model EULAG

The geophysical EULAG model (see [5], [8] for references) integrates Navier-Stokes equations using either EUlerian (flux form) or LAGrangian (advective form) numerics. The analytic formulation of EULAG assumes nonhydrostatic equations of motion in Bousinesq and anelastic approximations. The anelastic approximation removes the speed of sound from the Courant-Friedrichs-Lewy (CLF) stability criteria of an explicit integration (i.e. sound-proof equations [9]).

CRM may involve a broad range of microphysical parameterizations depending on the spatial scales of interest and the range lof physical processes involved. Parameterizations implemented in earlier versions of EULAG included the abbreviated bulk precipitating thermodynamics [10], with a relatively small number of condensate types, but also sophisticated droplet-size-resolving bin microphysics [12]. Here we present comparison of two microphysical parameterizations: a single moment bulk warm-rain scheme [8] and a bin-based [11] microphysical schemes.

2.1 Bulk Microphysics Scheme

The bulk scheme accounts for the effects of phase-change representing condensation of water vapor to form cloud condensate (C_d), the "autoconversion" (i.e., initial source of precipitation, A_p), growth/evaporation of precipitation due to diffusion of water vapor (E_p), and growth of precipitation due to accretion of cloud condensate (C_p). The governing thermodynamic equations can be written

$$\frac{D\theta}{Dt} = \frac{L\theta_e}{c_p T_e}(C_d + E_p) + D_\theta \tag{1}$$

$$\frac{Dq_v}{Dt} = -C_d - E_p + D_{q_v} \tag{2}$$

$$\frac{Dq_c}{Dt} = C_d - A_p - C_p + D_{q_c} \tag{3}$$

$$\frac{Dq_p}{Dt} = \frac{1}{\overline{\rho}}\frac{\partial}{\partial z}(\overline{\rho} V_T q_p) + A_p + C_p + E_p + D_{q_p} \tag{4}$$

where θ is the potential temperature; q_v, q_c, and q_p are water vapor, cloud condensate, and precipitation mixing ratios, respectively; ρ is the base-state anelastic density profile; D terms represent sources and sinks of model variables due to processes not directly represented in Eq: (1-4) (such as surface fluxes or turbulent transport); the subscript e (at θ_e and T_e) refers to the ambient (environmental) profiles; L and c_p and V_T denote the latent heat of condensation, the specific heat at constant pressure, and terminal velocity of the precipitation.

2.2 Bin Microphysics Scheme

The multiscale approach of the newly developed warm-rain bin-based microphysics include effects of entrainment and mixing on droplet size distribution [12]; turbulent fluctuations on droplet growth [1]; diffusion and collision-coalescence [13]; and precipitation formation [14]. The bin scheme solves the equation for the spectral density function f for the concentration of droplets N at a spatial location x, and in the droplet radius interval $(r, r + dr)$:

$$f(r, x) = \frac{DN(x)}{Dr} \tag{5}$$

$$\frac{\partial f}{\partial t} + \frac{1}{\overline{\rho}}\nabla \cdot (\overline{\rho}[\mathbf{v} - \mathbf{k}v_t(r)]f) + \frac{\partial}{\partial r}\left(\frac{dr}{dt}f\right) = \left(\frac{\partial f}{\partial t}\right)_{AC} + \left(\frac{\partial f}{\partial t}\right)_{CC} + D_f \tag{6}$$

The second and third terms of the left-hand side of (6) represent the advection of droplets ion the physical space (including droplet sedimentation) and their diffusional growth, respectively. The terms "AC", "CC", and D_f on the right-hand side represent sources due to activation (AC), the collision/coalescence (CC), and parameterized turbulent transport. The activation term is relevant only for

the bin corresponding to the initial droplet radius of 1 μm. The coalescence term includes two processes: a source of droplet formation in a particular bin owing to collisions of two droplets from different bins, and a sink representing collisions of droplets from this bin with all other droplets. The terminal fall speed $v_t(r)$ is calculated using the approximation in [15]. The discrete system consists of n bins (or classes) of droplets of the radius width $\Delta r(i)$, with $N(i)$ concentration, and spectral density functions $f(i) = N(i)/\Delta r(i)$ in (i) bin. The grid in radius space is a combination of linear and exponential, with mean radius r $[\mu m]$ given by $r(i) = 0.25(i-1) + 10^{0.055(i-1)}$. The linear spacing minimizes spectral dispersion during condensational growth of small droplets (< 15 μm), while the exponential spacing is required for larger radii to provide a stretched grid incorporating drizzle/rain sizes.

2.3 Numerical Approximation

The model prognostic equations are integrated using second-order nonoscillatory forward-in-time (NFT) approach [16] on an unstaggered grid. The boundary value elliptic problem for pressure perturbations is solved using a generalized conjugate residual - preconditioned nonsymmetric Krylov subspace GCR algorithm (see [17] for details). The sub-grids terms D are evaluated explicitly to the first-order accuracy inside the sub-grid scale turbulence model.

The water substance variables enter the dynamics of moist air and clouds only through the buoyancy term in the vertical momentum equation, and, except for the buoyancy, the equations of motion are virtually the same as for the dry dynamics. To accommodate a broad range of coarse spatial resolutions, the disparity between the timescales of the fluid flow and short timescales associated with phase-change processes and precipitation fallout, the approach based on the method of averages is employed [8]. In such approach the fast processes are evaluated with adequately small time steps (and lower accuracy) over the large time step of the dynamic model. This approach allows for stable integrations when cloud processes are poorly resolved and it converges to the explicit formulation (standard in cloud models) as the resolution increases.

3 Parallel Implementation

The EULAG code uses massively-parallel message-passing, where the interprocessor communication employs either Message Passing Interface (MPI) or Shared Memory (SHMEM) parallel libraries. The traditional parallelization strategy adopted initially in EULAG for the three-dimensional computations accounts for characteristic anisotropy of the media and favors checkerboard like 2D horizontal domain decomposition while maintaining the vertical direction unpartitioned. Such approach allowed to run quite efficiently on TeraScale parallel architectures applying hundred and thousands of processors [5]. Recently, a fully 3D domain decomposition was developed [6] where the domain is partitioned in each

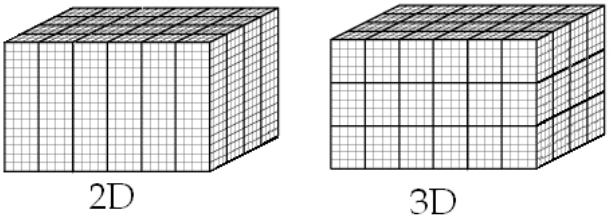


Fig. 1. Model parallel domain decomposition methods: checkerboard-like 2D domain decomposition in the horizontal direction in the left panel, and the newly developed 3D decomposition in the right panel

physical direction (Fig. 1). Such approach increases model scalability on PetasScale systems with ten and hundred thousands of processors. In both decompositions, a subdomain is assigned statically to a single processor responsible for calculations in this subdomain. The communication between processors requires assigning logical relationships between processors to track processor numbers and relation with their neighbors. Fig. 2 presents the logical redistribution of processors in both decompositions: 9 processors in 2D horizontal configuration (3x3) vs 27 processors in 3D configuration (3x3x3). Performing discrete differentiations require interprocessor communication and exchange information at the border of the sub-domains. For this purpose, a variable size (ih) area (i.e., the halo or ghost cell, shaded areas in Fig.2) is designated and it contains information sent by the neighbor processors. In the 3D decomposition, the communication cost (i.e. the ratio of the number of grid points within the halo area to the number of iner grid points on the processor) is smaller than in the 2D decomposition, which increases model performance.

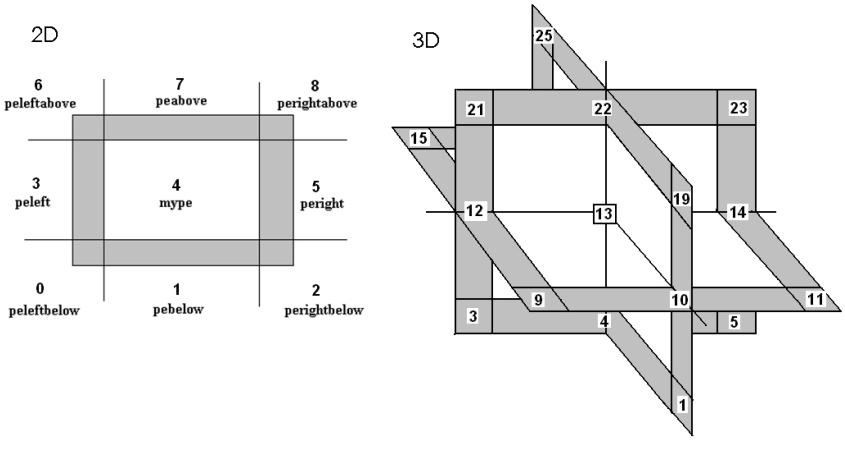


Fig. 2. Processor near neighbor interconnection for 2D and 3D decompositions

All model algorithms (except the GCR preconditioner) are fully parallelizable in the vertical direction. The line relaxation preconditioner is based on the sequential tridiagional Thomas algorithm which is naturally better suited for the 2D domain decomposition. In the 3D decomposition a pipelined version of Thomas algorithm (PTA) is developed [18] which minimizes communication in vertical direction by splitting the horizontal subgrids into portions that quickly fill up the pipeline of processors in vertical using single gather/scatter operations.

4 Benchmarks Outline

Three computer systems were used in the scalability experiments. The IBM BlueGene/L (BG/L) "Frost" is a massively parallel architecture used by the National Center for Atmospheric Research (NCAR) and the University of Colorado (CU). It is configured with 4 racks (8192 cores) reaching 22.9 Teraflops of peak performance. A single node of BG/L consists of two PowerPC 700 MHz processors and 512 MB of RAM memory, working in co-processor and virtual modes. In the coprocessor mode, one processor performs all computations while the other one is responsible for handling communication (e.g. reduction operations). In the virtual mode, both CPUs are running computations and share node resources. The point-to-point interconnection network is organized into 3D mesh or torus of 64x32x32 dimension, with bandwidth of 154 MB/s/link, and latency 3.35 μs. The global reduction and broadcast operations are performed with fast collective network with bandwidth 337 MB/s and small latency 2.5 μs. The Cray XT4 "Franklin" at the National Energy Research Scientific Computing Center (NERSC) is a massively parallel system with 38640 compute cores (9532 quadcore 2.3 GHz AMD-Opteron processors), 78 TB of memory, and 436 TB of parallel scratch disk space. The nodes are arranged in a 3D torus topology of dimension 21x16x24. The system is capable of providing 356 Teraflops of computational power. The Cray XE6 "Hopper II" system has 6392 nodes, (153,408 cores) with 217 TB of memory and 2 PB of disk space, and reaches peak performance of 1.288 Petaflops. The compute nodes are connected via Cray's custom high-bandwidth, low-latency network. The interconnection between nearby nodes is in the form of a "mesh", whereas the "edges" of this mesh form a 3D torus of 17x8x24 dimension.

4.1 Precipitating Thermal with Explicit Bulk Model

In the first computational benchmark we use explicit bulk microphysics to simulate 3D moist precipitating thermal evolving in the sheared environment. The default model domain consists of [256x256x160] grid points with uniform 40 m gridlength and time step of 0.5 s. The initial spherical buoyancy perturbation of the radius of 250 m is prescribed in the domain center, 1500 m above the ground. Boundary conditions are periodic in x, y and rigid-lid in z. Fig. 3 shows the development of a cloudy thermal. Initially at rest, the moist thermal starts to rise due to the imposed buoyancy excess. The accompanying adiabatic cooling results

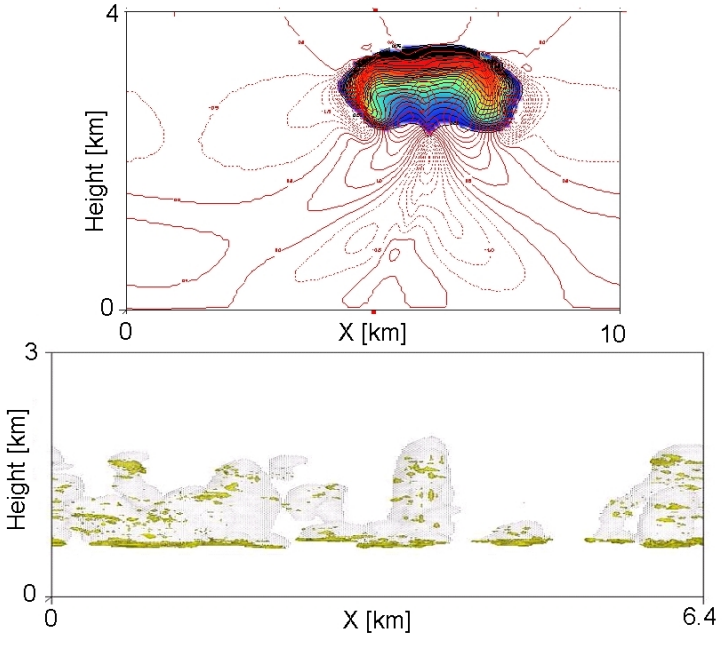


Fig. 3. Upper panel: the cloud water mixing ratio (color) and vertical velocity field with contour intervals 0.5 $[gkg^{-1}]$ and 0.5 $[ms^{-1}]$ respectively. Lower panel: Snapshot of cloud water mixing ratio (shaded isosurfaces) of value $qc = 0.05$ $[gkg^{-1}]$ and the activation tendency larger than 1 $[(mgs)^{-1}]$ (yellow isosurfaces).

in the condensation and eventually leads to the precipitation development. At later times, evaporation of rain leads to the development of a downdraft below the thermal.

4.2 Shallow Stratocumulus Simulations with Bin Microphysics

In the second test case, the experiments are set on a uniform 10 m grid spacing for a marine ice-free shallow cumulus simulations based on the Barbados Oceanographic and Meteorological Experiment (BOMEX) case [7]. The default model grid size uses [256x256x180] points, the time step of the simulation is $dt = 1.5$ s, and the whole simulation last 6 hours. The detailed bin-cloud microphysics is implemented, with 72 bins covering droplet sizes between 1 and 6000 μm. The drops with radius less than 40 μm are assumed to be cloud droplets, those greater than this size are assumed to be drizzle/rain. Lower panel on Fig. 3 shows snapshot of the steady-state trade wind shallow non-precipitating convection after 6 hour of simulations. The 1.5-km-deep trade-wind convection layer overlays 0.5 km deep mixed layer near the ocean surface and is covered by 500 m deep trade wind inversion layer. The cloud cover is about 10and quasi-steady conditions are maintained by the prescribed large-scale subsidence, large-scale moisture advection, surface heat fluxes, and radiative cooling.

5 Parallel Performance

In the 3D domain decomposition, a smaller amount of the halo information is exchanged compared to the 2D case. On the toroidal network, the 3D sub-domains are better mapped to virtual processor configuration, which results in a more localized data structure and less complicated communication while performing global operations (e.g. fast reduction and broadcast). This is especially important on IBM BG/L where separate network supports collective communications. The advantage of using the 3D decomposition is seen in Fig. 4. The results

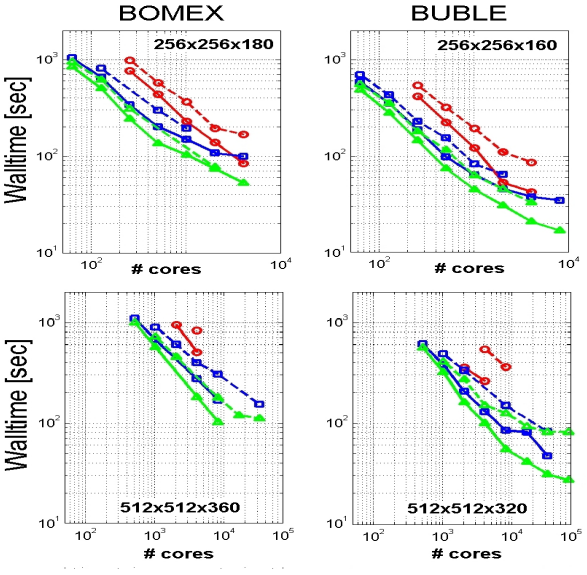


Fig. 4. Strong scalability results with full model physics. The red, blue, and green lines shows results from IBM BG//L, CRAY XT4 and Cray XE6 respectively, the dashed lines represent 2D decomposition, the continuous lines 3D decomposition. Upper and lower panels show default and double resolution problems respectively.

presented for different test cases on IBM BG/L show that, in both test cases, the 3D decomposition preserves perfect scalability for approximately four times larger number of computing nodes. The lower performance of the PTA algorithm requires that, for the largest number of processors, the number of subdomain in vertical must be smaller than in the horizontal. Model performance is not optimal in such a case as previously shown in [6]. Combined with the increasing communication cost for the largest number of cores, this leads in saturation of scalability. Such saturation effects related to the increase of the communication cost at the largest number of cores are commonly observed in other geophysical models, see [19], [21], [20] for references. However, in the current simulations, the low resolution runs show that the model is scalable up to 8192 cores in thermal

case and 4096 cores in BOMEX case. The double resolution runs extended the scalability to 65,536 and 32,768 cores respectively. The microphysical scheme follows the scalability pattern of the whole algorithm. The average overhead due to use of bulk microphysics ranges depending on application between 11 and 17 %, while the cost of using the bin scheme ranges between 50-54 %.

It was also observed that the model performance is sensitive to work-load imbalances due to the logical mapping pattern between nodes. The different scalability achieved at the tested systems may be related to improper mapping pattern between logical distribution of model sub-domain elements and compute nodes organized on toroidal interconnection networks. In the current experiments, only the default mapping algorithms were used due to limited computer resources, and further tests of this aspect are required.

6 Summary

We have demonstrated the applicability and efficiency of the cloud resolving model EULAG on contemporary high performance massively parallel systems. Performance has been assessed by conducting series of scalability experiments with two different microphysical models. The newly developed 3D model domain decomposition increases the scalability for up to 4 times more cores than the 2D decomposition. The model with 3D domain partitioning scales well up to 8K processors on the IBM BG/L and Cray XT4 and XE6 architectures for small-size problems. The scalability increases up to 32-64 thousands of cores when the size of the problem is doubled. The model performance is sensitive to work load imbalance impacted by the logical mapping pattern between model sub-domain elements and compute nodes organized on toroidal interconnection networks, which calls for further investigations in this area.

Acknowledgments. This work has been supported by the NSF through grants OCI-0904534 and OCI-0904449 and in part by the DE-FG02-08ER64535 DOE award. 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.

References

1. Wang, L.-P., Ayala, O., Rosa, B., Grabowski, W.W.: Turbulent collision efficiency of cloud droplets. New Journal of Physics 10, 075013 (2008)
2. Grabowski, W.W.: Representation of turbulent mixing and buoyancy reversal in bulk cloud models. J. Atmos. Sci. 64, 3666–3680 (2007)
3. Grabowski, W.W.: An improved framework for superparameterization. J. Atmos. Sci. 61, 1940–1952 (2004)
4. Kuang, Z.M., Blossey, P.N., Bretherton, C.S.: A new approach for 3D cloud resolving simulations of large-scale atmospheric circulation. Geoph. Res. Letters 32, L02809 (2005)

5. Prusa, J.M., Smolarkiewicz, P.K., Wyszogrodzki, A.A.: EULAG, a computational model for multiscale flows. Comput. Fluids 37, 1193–1207 (2008)
6. Piotrowski, Z.P., Wyszogrodzki, A.A., Smolarkiewicz, P.K.: Towards petascale simulation of atmospheric circulations with soundproof equations. Acta Geoph. 59, 1294–1311 (2011)
7. Wyszogrodzki, A.A., Grabowski, W.W., Wang, L.-P.: Activation of cloud droplets in bin-microphysics simulation of shallow convection. Acta Geoph. 59, 1168–1183 (2011)
8. Grabowski, W.W., Smolarkiewicz, P.K.: A multiscale anelastic model for meteorological research. Mon. Wea. Rev. 130, 939–956 (2002)
9. Smolarkiewicz, P.K., Szmelter, J.: A nonhydrostatic unstructured-mesh soundproof model for simulation of internal gravity waves. Acta Geoph. 59, 1109–1134 (2011)
10. Grabowski, W.W.: Toward cloud resolving modeling of large-scale tropical circulations: A simple cloud microphysics parameterization. J. Atmos. Sci. 55, 3283–3298 (1998)
11. Morrison, H., Grabowski, W.W.: Comparison of bulk and bin warm rain microphysics models using a kinematic framework. J. Atmos. Sci. 64, 2839–2861 (2007)
12. Andrejczuk, M., Grabowski, W.W., Malinowski, S.P., Smolarkiewicz, P.K.: Numerical simulation of cloud-clear air interfacial mixing: Effects on cloud microphysics. J. Atmos. Sci. 63, 3204–3225 (2006)
13. Grabowski, W.W., Wang, L.-P.: Diffusional and accretional growth of water drops in a rising adiabatic parcel: Effects of the turbulent collision kernel. Atmos. Chem. Phys. Discuss. 8, 14717–14763 (2008)
14. Wang, L.P., Grabowski, W.W.: The role of air turbulence in warm rain initiation. Atmos. Sci. Letters 10, 1–8 (2009)
15. Beard, K.V.: Terminal velocity and shape of cloud and precipitation drops aloft. J. Atmos. Sci. 33, 851–864 (1976)
16. Smolarkiewicz, P.K., Szmelter, J.: An MPDATA-based solver for compressible flows. Int. J. Numer. Meth. Fluids 56, 1529–1534 (2008)
17. 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. 203–220 (2004)
18. Povitsky, A.: Parallelization of pipelined algorithms for sets of linear banded systems J. Parallel Dist. Com. 59, 68–97 (1999)
19. Dennis, J.M., Spotz, W.F., St.-Cyr, A., Taylor, M.A., Thomas, S.J., Tufo, H.: High-resolution mesh convergence properties and parallel efficiency of a spectral element atmospheric dynamical core. Int. J. High-Performance Comp. Appl. 19, 225–235 (2005)
20. Schättler, U., Fuhrer, O.: Preparing the COSMO-model for future HPC architectures. In: 32nd EWGLAM and 17th SRNWP Meetings, Exeter, UK, October 4-7 (2010)
21. Michalakes, J., Hacker, J., Loft, R., McCracken, M., Snavely, A., Wright, N.J., Spelce, T., Gorda, B., Walkup, R.: WRF nature run. In: Proceedings of the 2007 ACM/IEEE Conf. on Supercomputing, Reno, USA, November 10-16 (2007)

Highly Efficient Parallel Approach to the Next-Generation DNA Sequencing

Jacek Blazewicz[1,2], Bartosz Bosak[3], Piotr Gawron[1], Marta Kasprzak[1,2], Krzysztof Kurowski[3], Tomasz Piontek[3], and Aleksandra Swiercz[1,2,⋆]

[1] Institute of Computing Science, Poznan University of Technology, Poznan, Poland
[2] Institute of Bioorganic Chemistry, Polish Academy of Sciences, Poznan, Poland
[3] Poznan Supercomputing and Networking Center, Poznan, Poland
aswiercz@cs.put.poznan.pl

Abstract. Due to the rapid development of the technology, next-generation sequencers can produce huge amount of short DNA fragments covering a genomic sequence of an organism in short time. There is a need for the time-efficient algorithms which could assembly these fragments together and reconstruct the examined DNA sequence. Previously proposed algorithm for de novo assembly, SR-ASM, produced results of high quality, but required a lot of time for computations. The proposed hybrid parallel programming strategy allows one to use the two-level hierarchy: computations in threads (on a single node with many cores) and computations on different nodes in a cluster. The tests carried out on real data of *Prochloroccocus marinus* coming from Roche sequencer showed, that the algorithm was speeded up 20 times in comparison to the sequential approach with the maintenance of the high accuracy and beating results of other algorithms.

Keywords: hybrid programming paradigm, *de novo* assembly.

1 Introduction

A molecule of *deoxyribonucleic acid* (*DNA*) can be found in every cell of a living organism. It carries genetic information necessary for its functioning. Despite the fact that DNA is similar for individuals from the same species, each individual has a unique DNA sequence. These differences in DNA cause different appearance of each individual but may also cause susceptibility to some diseases. Thanks to the knowledge of specific differences in DNA, a proper therapy can be applied to an ill patient. Thus, its proper decoding has a crucial meaning in all the aspects of the health issues.

DNA sequence is in the form of a double helix, where two strands are closely connected by hydrogen bonds according to specific rules. Each strand is composed of small molecules, called nucleotides, which differ in nitrogenous bases. Four different bases can be distinguished: adenine, cytosine, guanine and thymine,

⋆ Corresponding author.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 262–271, 2012.

and are abbreviated respectively as A, C, G, T. Reading a DNA sequence means reading the order of nucleotides (letters A, C, G, and T). It is sufficient to read only one strand, because the other one is complementary to it, which means that if there is 'A' in one strand, then there is 'T' at the same position in the other strand. Similarly, 'C' is always opposite to 'G'. The DNA assembly is a part of the process of reading a DNA sequence. In the DNA assembly problem short fragments of DNA (of length up to a few hundreds of nucleotides) are combined together in order to construct a longer sequence, e.g. a sequence of a whole chromosome of length ca 108 nucleotides. The set of short DNA fragments is the output of the preceding process: DNA sequencing. Originally, in DNA sequencing phase gel-based methods were used [15,16], which produced long fragments but with a lot of effort and time. A few years ago several techniques of *next-generation sequencing* were developed, which read huge amount of DNA fragments in parallel. Among next-generation sequencing systems the most known are *Roche* (454) *sequencer* [14], *Illumina sequencer* [1] and *SOLiD Applied Biosystems sequencer* [8]. Huge amounts of data force the implementation of time-efficient algorithms which optimize the memory usage.

DNA assembly problem is strongly NP-hard, because even its simplified version, shortest common superstring problem is strongly NP-hard [9] (cf. also [10]). In DNA assembly problem the aim is to reconstruct a sequence from shorter DNA fragments. Fragments may contain errors like insertions, deletions or mismatches, and may come from both strands of a DNA helix.

Several heuristic algorithms were developed, both for assembling DNA fragments coming from gel-based methods, e.g. [11,13,18], and for shorter fragments obtained with next-generation sequencers, e.g. [12,19]. Some methods are specialized for the special type of the sequencer, e.g. *Newbler assembler* is designed only for Roche sequencer [14]. The method developed previously by the authors of the current paper [2] proved to give as its output long reconstructed sequences of high quality, outperforming other available methods. The crucial point of the algorithm was long computation time, even in the case of parallel version [5].

The aim of the current paper was to develop an algorithm, which will maintain high accuracy of the previous approach but being much faster. The proposed hybrid strategy allows to use the two level hierarchy: computations in threads (on a single node with many cores) and computations on different nodes in a cluster. The applied parallel mechanism reduces computational time giving at the same time a high quality of the results.

A construction of the paper is as follows. Section 2 describes the algorithm for DNA assembly while Section 3 – the parallel version of the algorithm. In Section 4 results of the computational experiment are presented. The last section concludes the paper.

2 Short Reads ASseMbly (SR-ASM) Algorithm

SR-ASM algorithm [2] is based on operations on an overlap graph. Here, we modify slightly a concept of DNA graphs introduced in [4]. The vertices of the

overlap graph correspond to DNA fragments which are connected by arcs if there exists a feasible overlap between vertices. Next, in this graph a path(s) is searched for, which passes a maximum number of vertices. At the end, the sequence(s) is reconstructed from the path(s). Each step of the algorithm is described in details below.

Construction of the Overlap Graph. Every DNA fragment corresponds to one vertex in the graph. Additionally, for each vertex its complementary counterpart is created, which corresponds to the fragment coming from the other strand of the DNA helix, which is reverse and complementary to the original fragment. Next, the overlaps between pairs of fragments are calculated with the Smith-Waterman (SW) algorithm [17]. The algorithm aligns two fragments returning an error rate (the number of mismatches in the alignment) and a shift between fragments. The SW algorithm takes $O(kl)$ time, where k and l are the lengths of the fragments. The number of alignments, which need to be calculated, is $O(n^2)$, where n is the number of fragments. The most preferable would be to determine the alignment between each pair of the fragments (every one taken twice, as the original and the reverse and complementary version), but then the number of fragments goes up to a million. Most of the fragments do not overlap with each other in a satisfying manner, thus, determining the alignment is not necessary. The algorithm selects in a fast and intelligent way the pairs of fragments which possibly give high score of the alignment. In our algorithm, a heuristic was introduced, which searches for common substrings in pairs of fragments. Based on this, it selects pairs of promising fragments, i.e. the fragments which would overlap with high probability.

For each pair of the promising fragments the alignment is determined according to the SW algorithm. The score of the alignment gives an information about the number of errors (mismatches, insertions and deletions), but also about the shift between two fragments. If the shift and the error rate for two fragments is feasible, an arc is added to the graph which connects corresponding vertices.

The construction of the graph is composed of many independent operations, very suitable for distributing among parallel processes.

Searching for the Path in the Graph. In the overlap graph a path is searched for, which contains most of the vertices. (Here, the approach is very similar to the one constructing a DNA sequence in the SBH approach [3,6]) Optimally, it will be a Hamiltonian path. Unfortunately, due to errors present in the instance it is usually not possible. Another reason is that the heuristic used for construction of the overlap graph, selects just few pairs of fragments for further comparison, and many connections between vertices are not considered at all. In that case, the graph is quite sparse and might not be connected. Following this fact, many shorter paths are returned. Some vertices may be not used in any of the path, because they may represent contaminated fragments.

As the output of this step we obtain for each path an ordered set of vertices, for which respective fragments are tightly overlapping.

Printing the Consensus Sequence. In this step, the consensus sequences are constructed from the paths. Consecutive fragments (vertices) from each path are aligned together. The fragments which cover a certain position (letter) in the consensus sequence vote and the majority settle the proper letter (A, C, G or T). The consensus sequence of a path is called a *contig* (contiguous consensus sequence)

Again the problem of multiple alignment of the sequences is NP-hard, thus heuristic algorithm was applied here.

3 Parallelization of the Algorithm

The parallelism can be introduced to the aforementioned algorithm, especially in the first step. However, one should remember that the process of building the graph is composed of a few parts. Some parts cannot be parallelized at all and some to a satisfactory extent. Although the comparison of fragments with each other was efficiently performed by different threads (on different nodes), the stage of selecting fragments to be compared was implemented sequentially, due to high demand on the memory and relatively short computation time.

The second part of the algorithm (searching for paths in the graph) was also parallelized. This part is more complex and CPU-time consuming than the previous one. Searching for paths in a graph is, in general, very difficult to perform in a parallel way. Nevertheless, our method has partially solved this problem by running the same algorithm (on many nodes, in many threads) starting from different initial points. In each iteration every fragment is taken as the initial solution. The longest path is chosen and the arcs from the path are deleted from the graph.

This approach has two major disadvantages. First, each thread on every node requires full information about the graph, which dramatically increases requirements of memory and limits size of problem instances that can be processed. The second problem involves the implementation of searching method. Our solution offered the possibility of modifying the structure of the graph by adding some arcs during the phase of finding paths. If the algorithm finds out that there is no way to continue the process of extending the path from the last vertex, but there are some premises to continue the path, coming from the preceding vertices, additional alignments are calculated and if feasible – respective arcs are added. Possible extensions to the overlap graph are presented in Figure 1. Adding arcs to the graph implies possibility of desynchronizing information about the graph among different processes and could be the reason of receiving different results on different computing environments (various number of processes or threads, different computer architectures). The results obtained from the desynchronized data may be of lower quality. To avoid these situations, additional functionality was implemented - the *synchronization*. After every iteration of the algorithm, information about modifications in the graph is gathered from all processes and, if there are any differences, the graphs from each node are synchronized. In such a case, the iteration is calculated from the beginning. Synchronization ensures

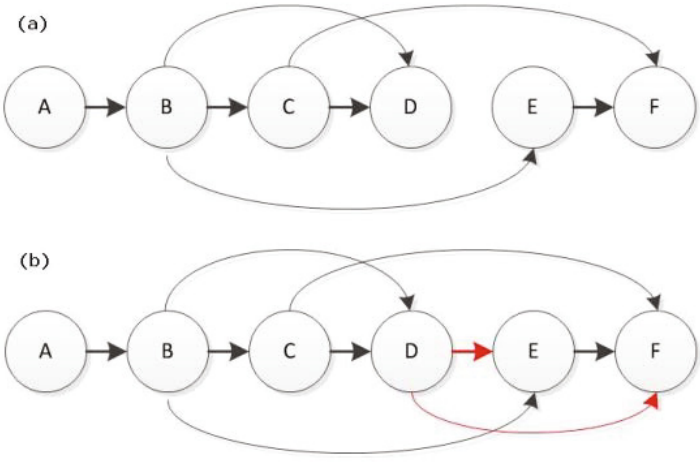


Fig. 1. An example of adding arcs in the overlap graph in the phase of searching for paths. (a) In the graph a path was found (A,B,C,D). The path cannot be extended because there is no arc outgoing from D, but some arcs from the predecessors of D were detected. (b) Scores of alignments between D and E, and D and F are calculated, and if they are feasible, the respective arcs are added to the graph. These arcs were not inserted in the first phase of the algorithm because the fragments (vertices) were not selected as 'promising pairs' (due to errors in fragments).

that the results obtained are the same as in the sequential version of the algorithm. At the same time, the synchronization leads to overhead time (single iteration may be computed a few times) and the user can decide to turn it off.

The third part of the algorithm (printing the consensus sequence) takes relatively little time and no attempt was made to parallelize it.

The implementation of parallelism in the program is based on a hybrid strategy introducing two level hierarchy: the communication between nodes in one cluster is realized by MPI library implementing the message passing paradigm, while the computations on a single node with many cores are parallelized utilizing POSIX threads. When the program is run on a single node, the first thread becomes a dedicated supervisor responsible for synchronization and distribution of the data as well as for doing sequential computations. The other available threads become worker units executing parallel computations. Similarly, in the case of using MPI, the first MPI process is designed to be a master, while the other MPI processes (slaves) do parallel computations.

In a typical scenario of running the SR-ASM program on a cluster, the process being an MPI master divides the tasks and distributes them among other MPI processes located on different nodes. Next, the MPI processes (dedicated supervisor threads) being a node-level coordinators, receive the data, once again divide the tasks and distribute them among the worker threads. After computations, the worker threads send results back to supervisor threads on theirs nodes and finally these threads response to the MPI master that gathers all the results.

The MPI communication is here both synchronous and asynchronous. The MPI master communicates with supervisor threads (MPI slaves) fully synchronously. It sends tasks and waits for results. When the results are obtained from one of the MPI slave processes, the master stores the data, generates next set of tasks and sends it to this process for further computations. On the other hand, on a single computing node level, worker threads communicate with the supervisor thread in an asynchronous way. Thanks to this approach, the computations and communication may be done simultaneously in order to improve efficiency of the whole process. Moreover, the implemented communication scheme utilizes queues to ensure that the next task to compute will be immediately available for processing after finishing the currently executed one. The performed tests confirm that all these mechanisms significantly improve an overall program scalability and performance.

4 Computational Tests

The scalability and performance tests were done on the Zeus cluster being a part of the Polish National Grid Initiative - PL-Grid Project. The cluster consists of several types of Intel-based nodes, which offer total computational power about 105 TF. The results presented in the paper are the outcome of running SR-ASM program on nodes with the following parameters:

Processor: Intel Xeon L5650 2666MHz (2 x 6 cores)
Memory: 24GB per node
Network Interface: Ethernet 1Gb/s
Operating System: Scientific Linux 5

The test bed used in the computational experiment consists of raw data produced at Joint Genome Institute. The data cover the whole genome of Prochlorococcus marinus bacteria of length 1.84 nucleotides [7]. The output of the sequencer contains over 300 000 DNA fragments, each approximately 100 nucleotides long. The sequencer also provides rates of confidence for every nucleotide.

As compared with its standard sequential version [2] the parallel SR-ASM improves the speed of the computations considerably. In fact, while the sequential version solved the analyzed instance of the problem in 255 min. (the first version of the algorithm published in [2] produced the results in 80 hrs., but since then it has been improved), its parallel version, even in the simplest case of the 1 node and 12 cores, needed 27 min only. What's more the parallel version is well scalable (to some extent).

Speedup of the whole algorithm with enabled and disabled synchronization is shown in Figure 2. One can easily see from this chart that the speedup for 24 cores is much lower than for 12 cores. This dramatic fall off is a result of changing the type of parallelism. For the first 12 cores, only one machine was used to compute results, while bigger number of cores was achieved by running SR-ASM on a cluster consisting of many 12-core nodes, what resulted in a higher communication overhead. Speedup of the version with synchronization turned

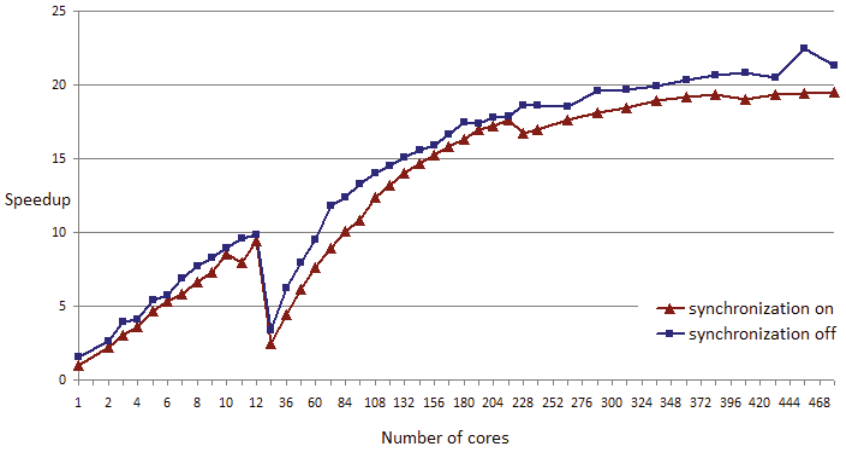


Fig. 2. Speedup of the parallel version of SR-ASM: version with synchronization turned on and synchronization turned off. Computations were performed on a computer cluster of 12-core nodes.

off varies between 30% (for the small number of cores) and 5% (for the big number of cores) as compared to the version with synchronization turned on. It is worth stressing that the parallel version of the algorithm with the synchronization turned on, maintains the high quality level for constructed sequences, already known for the sequential SR-ASM. As compared with other assembling algorithms: Newbler [14], Velvet [19], Phrap [11], Cap3 [12], the basic quality measures are favorable for SR-ASM. These measures include:

a) the *length* of the contig (for the three largest contigs),
b) the *coverage*, i.e. the percentage of the total length of the reconstructed sequence (genome), which is covered by the contig,
c) the *quality*, i.e. the similarity of a contig to a sequence of the genome; it was calculated by the Smith-Waterman algorithm.

The coverage and the similarity to a sequence of the reference genome can be calculated by comparing the obtained results with the reference genome. The reference genome of Prochlorococcus marinus is known and can be downloaded from Genbank (National Center for Biotechnology Information – NCBI).

The comparison of the results obtained for the considered instance are given in Table 1. Each algorithm is evaluated with the time of computations and the measures (mentioned above) for three largest contigs. Two versions of our algorithm are compared: with the synchronization turned on and off. Parallel SR-ASM with the synchronization on gives the same results in the quality terms as the sequential algorithm. When the synchronization is off, the algorithm does not update the information about new arcs in the graph (in the part of searching for paths). This may lead to nondeterministic results. Time of computations is

Table 1. The results of the computational experiment performed on the dataset of short fragments coming from the experiment of sequencing bacteria *Prochlorococcus marinus*. The length of the genome is 2Mbp. The number of fragments in the dataset is ca. $3 * 10^5$.

algorithm (time[s])	1st contig			2nd contig			3rd contig		
	length	coverage	quality	length	coverage	quality	length	coverage	quality
SR-ASM,s_on (781)	105951	5.66%	99.71	81990	4.38%	99.76	59714	3.19%	99.68
SR-ASM,s_off (714)	82209	4.39%	99.27	73071	3.90%	98.95	60475	3.23%	88.32
at JGI (-)	86108	4.60%	96.85	74315	3.97%	96.94	73941	3.95%	97.00
NEWBLER (-)	84798	4.53%	99.29	73192	3.91%	99.38	72818	3.89%	99.45
PHRAP (2067)	88729	4.74%	94.56	35379	1.89%	81.66	35192	1.88%	98.56
CAP3 (12376)	7862	0.42%	99.85	7113	0.38%	99.69	5990	0.32%	99.91
VELVET (328)	3182	0.17%	99.90	2808	0.15%	99.96	2621	0.14%	99.90

shorter than for SR-ASM with synchronization on, but the contigs are shorter, and often with lower quality. Thus, updating the information about the graph among the nodes seems to be very important. The results of other methods presented in Table 1 are obtained by assemblers from the Joint Genome Institute with an additional step of experts' finishing ('at JGI'), NEWBLER (assembler available with the Roche sequencer), PHRAP and CAP3 (well known publicly available assemblers), and VELVET (specialized algorithm for assembling short fragments). The executing time for NEWBLER and JGI were not available. All the methods except SR-ASM were tested sequentially. CAP3 and VELVET resulted in a great amount of short contigs but of very high quality. Thus, these methods are not likely to be used for *de novo* sequencing, but rather for re-sequencing, when the reference genome is known and good quality contigs are aligned to it to check for the difference between genomes of individuals. VELVET was the fastest method among all, while CAP3 the slowest. PHRAP obtained contigs shorter and of worse quality than the NEWBLER and JGI methods.

None of the algorithms found one contig covering the whole genome. This is usually the case, because the fragments are not uniformly distributed among the genome sequence. So, they result in disjoint contigs. Further order of the contigs can be determined e.g. on the basis of paired end fragments or by aligning contigs to a reference genome. If the reference genome is not known (this is the case of *de novo* sequencing) one may try to align contigs to a genome closely related to the examined organism.

5 Conclusions

The parallelization of SR-ASM algorithm satisfactorily speeded up the computations, especially in the case of multi-threading (almost linear speedup can be viewed for up to 12 cores in Fig. 2). The possible improvement of the algorithm could be realized on machines with more cores and shared memory to decrease the time needed for the communication and synchronization.

The results presented in Table 1 and in Figure 2 show, that although turning off synchronization may speed up the method (5-30% in comparison with the synchronization turned on), the results have worse quality. For a greater number of clusters the speedup goes down. This is a result of changing time distribution between the computation, communication and synchronization.

It can be seen, that the parallel version of SR-ASM maintained high quality of the results as for the sequential version, while the time was greatly speeded up. Thus, the proposed solution can be of practical interest for those who would like to sequence whole genomes, based on the massive data coming from next-generation sequencers.

Acknowledgements. We are very grateful to the anonymous referees for their helpful comments on how to improve the paper.

This work is partially funded by NCN grant (DEC-2011/01/B/ST6/07021), Polish NGI, and PL-Grid Project (POIG.02.03.00-00-007/08-00)

References

1. Bennett, S.: Solexa Ltd. Pharmacogenomics 5, 433–438 (2004)
2. Blazewicz, J., Figlerowicz, M., Gawron, P., Kasprzak, M., Kirton, E., Platt, D., Swiercz, A., Szajkowski, L.: Whole genome assembly from 454 sequencing output via modified DNA graph concept. Comput. Biol. Chem. 33, 224–230 (2009)
3. Blazewicz, J., Formanowicz, P., Kasprzak, M., Markiewicz, W.T., Weglarz, J.: DNA sequencing with positive and negative errors. J. Comput. Biol. 6, 113–123 (1999)
4. Blazewicz, J., Hertz, A., Kobler, D., de Werra, D.: On some properties of DNA graphs. Discrete Appl. Math. 98, 1–19 (1999)
5. Blazewicz, J., Kasprzak, M., Swiercz, A., Figlerowicz, M., Gawron, P., Platt, D., Szajkowski, L.: Parallel implementation of the novel approach to genome assembly. In: Lee, R., Muenchaisri, P., Dosch, W. (eds.) Proceedings of SNPD 2008, pp. 732–737. IEEE Computer Society, Los Alamitos (2008)
6. Blazewicz, J., Oguz, C., Swiercz, A., Weglarz, J.: DNA sequencing by hybridization via genetic search. Oper. Res. 54, 1185–1192 (2006)
7. Chen, F., Alessi, J., Kirton, E., Singan, V., Richardson, P.: Comparison of 454 sequencing platform with traditional Sanger sequencing: a case study with de novo sequencing of Prochlorococcus marinus NATL2A genome. In: Plant and Animal Genomes Conference (2006), http://www.jgi.doe.gov/science/posters/chenPAG2006.pdf
8. Fu, Y., Peckham, H.E., McLaughlin, S.F., Rhodes, M.D., Malek, J.A., McKernan, K.J., Blanchard, A.P.: SOLiD sequencing and Z-Base encoding. In: The Biology of Genomes Meeting. Cold Spring Harbour Laboratory (2008)

9. Gallant, J., Maier, D., Storer, J.: On finding minimal length superstrings. J. Comput. Sys. Sci. 20, 50–58 (1980)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco (1979)
11. Green, P.: Documentation for PHRAP. Genome Center, University of Washington, Seattle (1996)
12. Huang, X., Madan, A.: CAP3: a DNA sequence assembly program. Genome Res. 9, 868–877 (1999)
13. Kececioglu, J.D., Myers, E.W.: Combinatorial algorithms for DNA sequence assembly. Algorithmica 13, 7–51 (1995)
14. Margulies, M., Egholm, M., Altman, W.E., Attiya, S., Bader, J.S.: Genome sequencing in microfabricated high-density picolitre reactors. Nature 437, 376–380 (2005)
15. Maxam, A.M., Gilbert, W.: A new method for sequencing DNA. Proc. Natl. Acad. Sci. USA 74, 560–564 (1977)
16. Sanger, F., Nicklen, S., Coulson, A.R.: DNA sequencing with chain-terminating inhibitors. Proc. Natl. Acad. Sci. USA 74, 5463–5467 (1977)
17. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147, 195–197 (1981)
18. Waterman, M.S.: Introduction to Computational Biology. Maps, Sequences and Genomes. Chapman & Hall, London (1995)
19. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short reads assembly using de Bruijn graphs. Genome Res. 8, 821–829 (2008)

Parallel and Memory-Efficient Reads Indexing for Genome Assembly

Guillaume Chapuis, Rayan Chikhi, and Dominique Lavenier

Computer Science Department, ENS Cachan/IRISA, 35042 Rennes, France

Abstract. As genomes, transcriptomes and meta-genomes are being sequenced at a faster pace than ever, there is a pressing need for efficient genome assembly methods. Two practical issues in assembly are heavy memory usage and long execution time during the read indexing phase. In this article, a parallel and memory-efficient method is proposed for reads indexing prior to assembly. Specifically, a hash-based structure that stores a reduced amount of read information is designed. Erroneous entries are filtered on the fly during index construction. A prototype implementation has been designed and applied to actual Illumina short reads. Benchmark evaluation shows that this indexing method requires significantly less memory than those from popular assemblers.

1 Introduction

Until the emergence of next-generation sequencing (NGS) technologies, software for assembling genomes could process up to millions of long ($\sim 10^4$ bp) reads. Now, a typical genome assembly instance for a vertebrate genome consists of billions of short (100 bp) reads. Despite this technological shift, computational models for assembly are essentially based on constructing and simplifying a genome graph. However, graph-based models have inherent limitations that make them unpractical for assembly of NGS data. They require the construction of either a large string graph containing all the reads, or a de Bruijn graph containing all the k-length substrings (k-mers) of the reads. For human-sized genomes, the de Bruijn graph typically requires hundreds of gigabases of memory [9]. Nevertheless, NGS assembly tools rely on optimized implementations of these graph models. For instance, leading assembly programs have implemented efficient heuristics using a de Bruijn graph [9, 18]. For more details concerning these implementations, refer to a recent survey [10]. In a near future, larger eukaryotic genomes and meta-genomes will be sequenced at a faster pace than computational resources growth. Hence, new assembly models need to be developed to sustain the increasing rate of NGS technologies.

Several theoretical advances have been recently proposed to reduce the memory usage of graph-based assemblers. Simpson et al. implemented compression techniques (FM-index [6]) during construction of the string graph [15] at the expense of running time. Conway et al. used succinct bitmap structures [11, 13] to construct an immutable de Bruijn graph [5]. Distributed de Bruijn graph construction using a message passing interface have been implemented in several assemblers [7, 8, 16].

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 272–280, 2012.

Greedy assemblers use a different assembly strategy. Instead of constructing a genome graph, they repeatedly perform an extension procedure until branching is detected. Previous implementations of greedy assemblers used a prefix tree to store reads [17], which consumes significantly more memory than a de Bruijn graph. Recent optimized implementations use custom k-mer indexing structures for memory efficiency [1–3]. In particular, these implementation have been applied to complex mammalian genomes, demonstrating that greedy assemblers are not limited to genomes with low repeat content. Unlike de Bruijn graph assemblers, data structures used in greedy assemblers typically contain references to read sequences. Hence, efficient read indexing is necessary to keep memory usage low.

In the next section, we propose a parallel reads indexing procedure designed specifically for assembly. Two novel filtering methods are introduced to reduce memory usage: a procedure to remove erroneous k-mers on the fly, and a procedure to avoid referencing redundant reads. Finally, a prototype implementation is applied to real Illumina data to validate the method.

2 Methods

2.1 Distributed and Multi-threaded Indexing

A multi-threaded, multi-node procedure for reads indexing is proposed. A hash table is constructed, where the entries are k-mers, and the values are references to reads. Taking advantage of shared memory between threads, reads sequences are stored separately in memory, without redundancy within a node. Index construction is distributed among N nodes, and each node performs independent computations in parallel. Specifically, each node n is running T_n threads, each thread t_n constructs a separate sub-index $I(n, t_n)$. A binning method adapted from [16] assigns each k-mer to a unique sub-index. Let h be a k-mer hash value with perfect hashing [16]. The corresponding k-mer belongs to the sub-index $I(n, t_n)$ if:

$$\begin{cases} h \ mod \ N = n \\ h \ mod \ T_n = t_n \end{cases}$$

which ensures that each sub-index contains distinct k-mers. Each thread reads the entirety of the input data to construct its sub-index. When all the sub-indexes are constructed, an inexpensive merging phase yields the complete index. Hence, the indexing procedure always constructs the same complete index on different architectures. In the following, two algorithmic ingredients are described for parallel sub-index construction: k-mers filtering and reads indexing.

2.2 On-Line Parallel k-Mers Filtering

Memory efficiency is crucial when assembling NGS data. In many approaches, including the one proposed here, memory consumption is proportional to the

number of indexed k-mers. It is therefore important to filter out erroneous k-mers as early in the indexing process as possible. Erroneous k-mers are produced whenever the sequencing process makes a mistake during base calling. The abundance distribution $K_t^n(m)$ is defined as the number of k-mers seen exactly m times at indexing time t by node n. A key fact is that the hash function used above evenly distributes k-mers among sub-indexes. Hence, each $K_t^n(m)$ is identically distributed as the entire distribution $\sum_n K_t^n(m)$. This observation enables independent, parallel filtering for each sub-index. The superscript n is then omitted in the following.

A typical distribution of $K_t(m)$ at final time t is multimodal. A large number of k-mers occur only a few times: these are mostly sequencing errors. Assuming uniform sequencing coverage, the distribution of correct k-mers is a Gaussian mixture. The most abundant component is centered at the expected coverage of the target genome. Less abundant components are centered at multiples of the coverage, due to repeats in the genome. The proposed method consists in (i) detecting components corresponding to erroneous and correct k-mers as soon as they separate sufficiently from each other and (ii) finding an appropriate erroneous threshold (cut-off value). Every k-mer that has appeared fewer times than the erroneous threshold so far is then considered as an error and removed. This procedure could be extended to correct errors in reads, but it is outside the scope of the current indexing scheme.

Error Detection. The following two inequalities must be satisfied to trigger the filtering procedure. First, erroneous k-mers are identified by their abundance. Theorem 3 from [12] establishes that, under reasonable sequencing assumptions, an error is significantly less likely to appear $m+1$ times than m times. Thus, the abundance of erroneous k-mers peaks at $m = 1$ and has a strictly decreasing slope. The low end $m_{low}(t)$ is computed as the largest m that satisfies $K_t(m-1) > K_t(m)$ for $m \geq 1$. Then, the peak abundance $m_{high}(t)$ of correct k-mers is computed as the parameter at which the maximum value of $K_t(m)$ is attained for $m > m_{low}(t)$. Erroneous and correct k-mers are considered to be separated when:

$$m_{high}(t) - m_{low}(t) > r$$

where r is a user-defined resolution parameter. Second, to avoid the computational cost of filtering too soon or too often, a constraint is imposed on the amount of erroneous k-mers. Let S_{min} be a minimum amount (user-defined) of erroneous k-mers before the filtering process can be performed:

$$\sum_{m=1}^{m_{low}} K_t(m) > S_{min}$$

Calculating the Cut-Off Value. During early filtering passes, a small fraction of correct k-mers still contributes to the erroneous component. Hence, removing the entire component at each filtering pass is not a sensible choice. An incrementing

value, defined as the *cut-off value*, is introduced to overcome this problem. All k-mers of abundance lower than the cut-off value are removed by the filtering procedure, others are kept. Formally, a threshold m_{solid} is defined as the number of occurrences below which a k-mer is considered a potential error. All k-mers over this threshold at the end of the indexing phase are *solid* k-mers. Let $tReads$ and $nReads(t)$ be the total number of reads in the input file and the number of reads processed at time t respectively. The cut-off value $F(t)$ is calculated according to the following formula:

$$F(t) = \lfloor \frac{m_{solid} \cdot nReads(t)}{tReads} \rfloor$$

2.3 Reads Indexing Structure

Each sub-index is populated independently with a filtered set of references to reads, given a filtering function designed for *de novo* assembly. The *extension* of a k-mer in a read is defined as the suffix immediately following the k-mer (e.g. for a read $r = uwv$ where w is a k-mer and u, v are arbitrary strings, v is the extension of w in r). We introduce a notion of redundancy between extensions. Let (v_1, v_2) be two extensions of the same k-mer, without loss of generality assume that the length $|v_1|$ is shorter than $|v_2|$. Two extensions v_1, v_2 are said to be t-redundant if the Hamming distance between their prefixes of length $|v_1|$ is lower than t. The *representative read spectrum* with similarity threshold t, noted $RRS(k, t)$, is defined for a set of input reads as follows:

(i) associate a set S_w to each solid k-mer w occurring in the reads
(ii) S_w discards all but one of the reads associated to t-redundant extensions. A read with the longest extension is kept, ties are broken arbitrarily.

Figure 1 shows an example of a representative reads spectrum. The reads sequences referenced by the RRS are stored separately. Practically, both a read and its reverse-complement are indexed. References to paired-end reads are explicitly made to the left or the right mate of the read.

In essence, this structure records a representative set of reads for each solid k-mer. Note that this indexing does not correct errors in read, but merely ignores errors in reads suffixes. Erroneous prefixes yield un-solid k-mers, hence these reads are not indexed in the structure. This property is well suited with Illumina reads as sequencing errors are known to mostly occur at read suffixes. Provided the sequencing coverage is high, errors in suffixes can be corrected at a later stage during a consensus phase. This justifies the arbitrary removal of other reads having equally long t-redundant extensions. To maximize the effectiveness of the structure for assembly, sequencing reads should contain solid k-mers corresponding to every position in the genome. Hence, either a high sequencing coverage or a low error-rate is required. Both criteria are typically met with recent Illumina sequencers.

For assembly, it can be verified that basic traversal of a string graph can be performed with this structure. The RRS acts as an incomplete inverted index

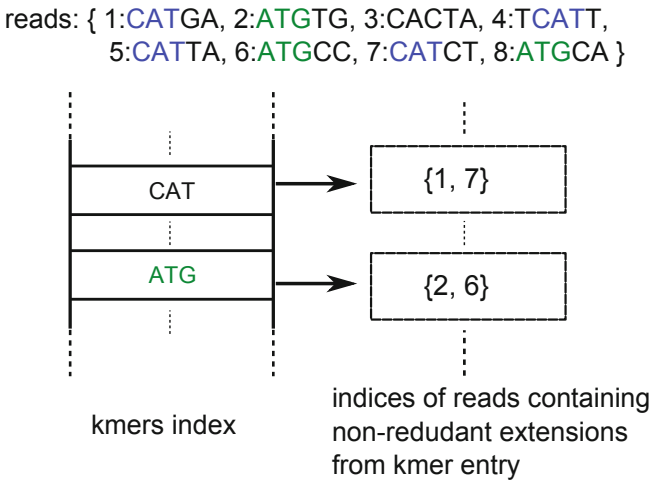


Fig. 1. The representative reads spectrum for a set of 8 reads with parameters $k = 3, t = 1$. Entries are solid k-mers from reads. Each k-mer is associated with a list of reads which extend the k-mer to the right. The extensions are filtered for t-redundancy. For instance, reads 4 and 5 are not indexed to the CAT entry because extensions T and TA are 1-redundant with respect to extension GA from read 1. Reverse complements of reads are also indexed, but are omitted in this figure.

for the reads. Specifically, in the string graph, out-neighbors of a read (i.e., other reads that overlap that read to the right) are retrieved from the RRS by querying each of the read k-mers. In-neighbors (left overlap) are equivalent to out-neighbors of the read reverse complement.

3 Results

We developed an implementation of the on-line k-mers filtering and the reads indexing algorithms, as part of the Monument assembler [4]. The implementation is tested on two actual sequence datasets from *R. sphaeroides* (SRA reference SRR034530) and *N.crassa* (all libraries from [14]) sequenced using the Illumina technology. The *R. sphaeroides* dataset (dataset 1) contains 46 million reads of length 36 bp. The *N.crassa* dataset (dataset 2) contains 320 million reads of average length 32 bp. Benchmarks were run on a 64-bit 8-cores machine with 66 GB of memory. In this implementation, read sequences are stored in memory on each node as an array of 2-bit encoded sequences. In the case of multi-nodes computation, $\frac{n}{4}$ bytes are redundantly stored per node, where n is the number of nucleotides in the reads. For the *R. sphaeroides* reads set, this amounts to 0.462 GB.

We first examined the effect of on-line k-mers filtering on the first dataset. To this end, only the abundance count is retained for each k-mer. A comparison against a k-mer counting without filtering is made in Figure 2. It is important to

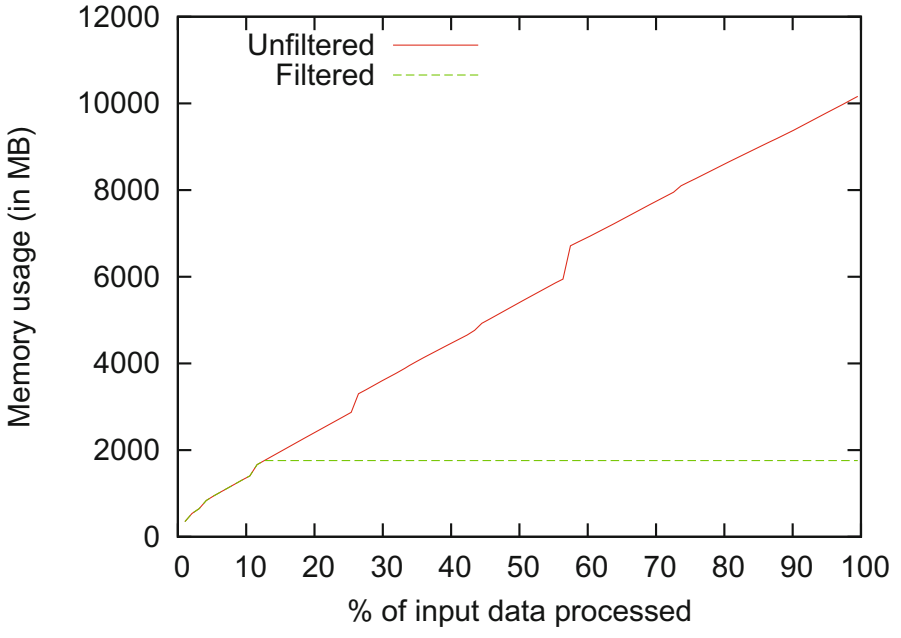


Fig. 2. Memory usage during the on-line k-mers filtering procedure, compared with un-filtered indexing. Dataset 1 is processed with parameters $m_{solid} = 10$, $r = 10$, $S_{min} = 10^7$ and using 1 thread. The first filtering pass is triggered at 11.6% of the dataset. Sporadic jumps in memory consumption correspond to resize operations of the hash table.

note that, when entries corresponding to erroneous k-mers are removed from the hash table, the allocated memory is not freed but is instead made available for new entries. There are 144 M k-mers in the dataset, only 4.5 M (3.1%) of which are correct. On-line filtering enabled to keep the number of k-mers in the hash table under 23 M at any time. We verified that 4,544,973 solid kmers are retrieved without filtering, compared to 4,464,256 (98.2%) solid kmers with filtering (solid threshold $m_{solid} = 10$). The difference of 80,717 k-mers corresponds to premature filtering of k-mers that would be solid if given enough time before filtering. Then, we computed the full indexing time for an increasing number of cores (Figure 3). Some constant over-head occurs as reads pre-loading is not parallelized.

We compared memory usage of indexing procedures from other popular ultra-short reads assemblers with our implementation. The Velvet assembler (version 1.1.03) and the SOAPdenovo assembler (version 1.05) are based on de Bruijn graphs and use graph simplification heuristics. SOAPdenovo is specifically optimized for memory efficiency, it discards reads and pairing information in the initial graph structure. Our implementation uses spectrum parameter $t = 4$, $S_{min} = 10^6$, $m_{solid} = 10$ and $r = 0$ for both datasets. All the assemblers are executed with k-mer size of 21. Only the indexing phase of assemblers were run (`pregraph` for SOAPdenovo, `velveth` for Velvet). Results are summarized in Table 1.

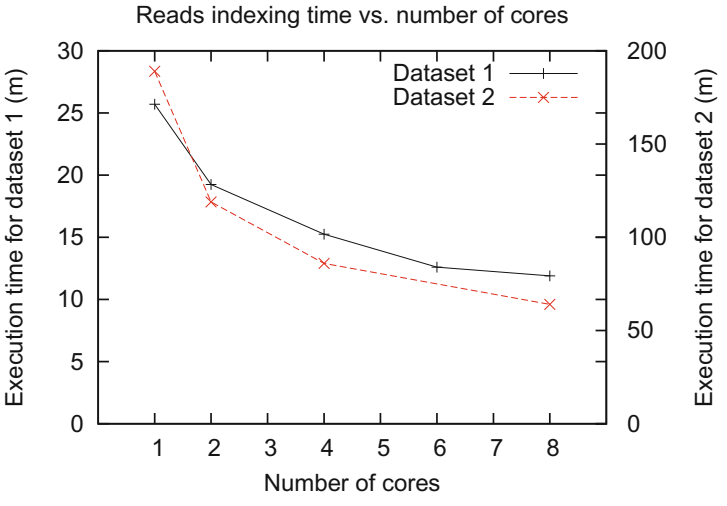


Fig. 3. Execution time of indexing for our implementation on datasets 1 and 2 using 1 node and 1 to 8 threads

The k-mers filtering step is essential in our method: complete indexing of Dataset 1 without k-mers filtering required 20.1 GB of memory. In terms of wall-clock time, these methods are comparable: for the largest dataset, SOAPdenovo and Monument completed indexing in respectively 41 and 64 minutes using 6 threads. In conclusion, our indexing scheme significantly reduces the memory bottleneck for assembly, with minor impact on parallel indexing time.

Table 1. Practical memory usage of indexing 46 M reads *from R. sphaeroides* (dataset 1) and 320 M reads from *N.crassa* (dataset 2) using Velvet, SOAPdenovo and Monument. Velvet exceeded the memory limit (66 GB) on the second dataset.

	Dataset	Monument	Velvet	SOAPdenovo
Peak memory (GB)	1	2.7	7.7	3.9
	2	15.3	-	31.4

We verified that this index permits assembly with comparable quality than existing methods. To this end, we implemented an assembly method which constructs contigs based on extensions recorded in the representative reads spectrum. We computed the following N50 values (contig length such that longer contigs produce 50% of the assembly) with respect to the same assembly size (that of Velvet, 4.30 Mbp). Our assembly of *R. sphaeroides* yields 4.28 Mbp of contigs with a N50 value of 1.49 kbp. In comparison, we executed Velvet with similar k-mer size, which yields 4.30 Mbp of contigs with a N50 value of 1.41 kbp. In terms of

running times, Velvet assembly phase (`velvetg`) executed in 2 min 46 sec CPU time, whereas our parallel string graph traversal required 1 min 42 sec CPU time per thread using 6 threads.

4 Discussion

A novel method is proposed for multi-nodes, multi-threaded reads indexing. It introduces two filtering techniques for memory efficiency: on-line removal of erroneous k-mers and on-line indexing of only representative reads. To our knowledge, this is the first read index that provides in-memory $O(1)$ access to overlaps between reads, full read sequences and also pairing information. This novel indexing method allows to design assembly algorithms which overcome several of the performance issues inherent to graph-based assembly. For instance, memory usage is lowered as no graph structure is constructed. Independent indexing of sub-indexes allows for embarrassingly parallel and distributed computation. As implemented in our prototype, the read index is constructed using significantly less memory than recent, optimized implementations of de Bruijn graphs with comparable indexing time.

These results can also be used to reduce memory usage of single-threaded greedy assemblers. Usually, the same data structure is used to construct the final reads index and then access it during assembly. However, greedy assemblers typically uses an immutable index. Hence, taking advantage of immutability, one can focus on designing a more compact representation of the reads index once it is fully constructed. For memory efficiency, sub-indexes can be simply constructed one at a time. In our index, the hash table can be replaced by a succinct rank/select data structure [11] to represent entries, and a simple array containing fixed-length lists of representative reads. The memory overhead of such structure becomes negligible as no pointer is used. A preliminary experiment shows that the entire index of dataset 2 is represented in only 4.2 GB of memory with this method.

References

1. Ariyaratne, P.N., Sung, W.: PE-Assembler: de novo assembler using short paired-end reads. Bioinformatics (December 2010)
2. Boisvert, S., Laviolette, F., Corbeil, J.: Ray: Simultaneous assembly of reads from a mix of High-Throughput sequencing technologies. Journal of Computational Biology, 3389–3402 (2010)
3. Chapman, J.A., Ho, I., Sunkara, S., Luo, S., Schroth, G.P., Rokhsar, D.S.: Meraculous: De novo genome assembly with short Paired-End reads. PloS One 6(8), e23501 (2011)
4. Chikhi, R., Lavenier, D.: Localized genome assembly from reads to scaffolds: practical traversal of the paired string graph. Algorithms in Bioinformatics, 39–48 (2011)
5. Conway, T.C., Bromage, A.J.: Succinct data structures for assembling large genomes. Bioinformatics (2011)

6. Ferragina, P., Manzini, G.: Indexing compressed text. Journal of the ACM (JACM) 52(4), 552–581 (2005)
7. Jackson, B., Schnable, P., Aluru, S.: Parallel short sequence assembly of transcriptomes. BMC Bioinformatics 10(suppl. 1), S14 (2009)
8. Kundeti, V., Rajasekaran, S., Dinh, H., Vaughn, M., Thapar, V.: Efficient parallel and out of core algorithms for constructing large bi-directed de bruijn graphs. BMC Bioinformatics 11(1), 560 (2010)
9. Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., Wang, J.: De novo assembly of human genomes with massively parallel short read sequencing. Genome Research 20(2), 265–272 (2010), http://genome.cshlp.org/content/20/2/265.abstract
10. Miller, J.R., Koren, S., Sutton, G.: Assembly algorithms for next-generation sequencing data. Genomics (2010)
11. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. Arxiv preprint cs/0610001 (2006)
12. Peng, Y., Leung, H.C.M., Yiu, S.M., Chin, F.Y.L.: IDBA – A Practical Iterative de Bruijn Graph De Novo Assembler. In: Berger, B. (ed.) RECOMB 2010. LNCS, vol. 6044, pp. 426–440. Springer, Heidelberg (2010)
13. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms (TALG) 3(4), 43–es (2007)
14. Shea, T., Williams, L., Young, S., Nusbaum, C., Jaffe, D., MacCallum, I., Przybylski, D., Gnerre, S., Burton, J., Shlyakhter, I., Gnirke, A., Malek, J., McKernan, K., Ranade, S.: ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. Genome Biology 10(10), R103 (2009), http://genomebiology.com/2009/10/10/R103
15. Simpson, J.T., Durbin, R.: Efficient construction of an assembly string graph using the FM-index. Bioinformatics 26(12), i367 (2010)
16. Simpson, J.T., Wong, K., Jackman, S.D., Schein, J.E., Jones, S.J., Birol, I.: ABySS: a parallel assembler for short read sequence data. Genome Research 19(6), 1117–1123 (2009)
17. Warren, R.L., Sutton, G.G., Jones, S.J.M., Holt, R.A.: Assembling millions of short DNA sequences using SSAKE. Bioinformatics 23(4), 500–501 (2007), http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/4/500
18. Zerbino, D.R., Birney, E.: Velvet: Algorithms for de novo short read assembly using de bruijn graphs. Genome Research 18(5), 821–829 (2008), http://genome.cshlp.org/content/18/5/821.abstract

Parallel Software Architecture for Experimental Workflows in Computational Biology on Clouds

Luqman Hodgkinson[1,2,3], Javier Rosa[1], and Eric A. Brewer[1]

[1] Computer Science Division, University of California, Berkeley, USA
[2] Center for Computational Biology, University of California, Berkeley, USA
[3] International Computer Science Institute, Berkeley, CA 94720, USA
{luqman,javirosa,brewer}@cs.berkeley.edu

Abstract. Cloud computing opens new possibilities for computational biologists. Given the pay-as-you-go model and the commodity hardware base, new tools for extensive parallelism are needed to make experimentation in the cloud an attractive option. In this paper, we present EasyProt, a parallel message-passing architecture designed for developing experimental workflows in computational biology while harnessing the power of cloud resources. The system exploits parallelism in two ways: by multithreading modular components on virtual machines while respecting data dependencies and by allowing expansion across multiple virtual machines. Components of the system, called elements, are easily configured for efficient modification and testing of workflows during ever-changing experimentation. Though EasyProt, as an abstract cloud programming model, can be extended beyond computational biology, current development brings cloud computing to experimenters in this important discipline who are facing unprecedented data-processing challenges, with a type system designed for proteomics, interactomics and comparative genomics data, and a suite of elements that perform useful analysis tasks on biological data using cloud resources.

Availability: EasyProt is available as a public abstract machine image (AMI) on Amazon EC2 cloud service, with an open source license, registered with manifest easyprot-ami/easyprot.img.manifest.xml.

Keywords: parallel architectures, scientific workflows, cloud computing.

1 Introduction

Cloud computing is set to change the way that bioinformatics research is conducted, improving the cost-effectiveness of massive computations [1]. Cloud computing provides an increasingly attractive alternative for computational biologists as datasets are increasing in size faster than desktop computers are increasing in capacity [1]. With its pay-as-you-go model, new computational tools are needed to make cloud computing an attractive option for scientists conducting computational experiments, to balance speed of applications and development with cost. A primary advantage, and challenge, of cloud computing

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 281–291, 2012.

is its ability for horizontal expansion [2]. Single virtual machines typically have several virtual cores while the power of each core remains relatively constant, and multiple virtual machines can be launched on demand.

The scientific workflow management system [3] is a framework that can be usefully adapted to make cloud computing attractive to experimenters in the computational sciences. In a scientific workflow, tasks are connected in a directed graph representing data dependencies, and data flows in parallel along the edges. Scientific workflows generalize ideas from pioneering projects on configurable services [4,5,6]. The strength of the scientific workflow as a cloud programming model is its ability to harness resources as they expand horizontally in the cloud. This computing development model can be easily built on top of Infrastructure as a Service [7] clouds, providing an attractive development environment for computational biologists.

There have been attempts to extend scientific workflow software into the realm of cloud computing [8], the most notable of which is Pegasus [9,10]. Pegasus is designed for large stable scientific applications, and is an excellent choice for this use case. Custom workflows using cloud computing have been designed with other scientific workflow software including Taverna [11] and Kepler [12]. However there remains a need for a programming framework that allows cost-effective and fast reconfiguration by programmers during scientific experimentation in the cloud. Very recently, a workflow system, Conveyor [13], was released that shares the goal of EasyProt for a programming model that allows fast configuration during ever-changing experimentation. However, Conveyor does not offer full support for cloud computing, is not released as an AMI, and requires substantial local installation.

EasyProt is workflow software for the programmer. EasyProt uses a graph configuration language, modelled after Click [5], that makes it fast and easy to create many experimental workflows on the same elements. Like Conveyor, EasyProt takes advantage of parallelism available in multicore virtual machines by using a multithreaded approach. EasyProt uses message-passing protocols that can pass arbitrary Java objects, allowing it to scale horizontally, an important feature for cloud computing applications [2]. Several message-passing protocols are part of EasyProt, including broadcast and round-robin message-passing, and new protocols can be programmed directly by the element designer. EasyProt supports a data cache for versioning and provenance that is separable from the workflows. In case of failure of a workflow for any reason, intermediate cached results are available that can be used for a modified workflow using data from the cache. The EasyProt system follows modular design, making it easy to modify workflows or to share workflows with other users, and to incorporate new algorithms, new web services, and support for new forms of data as they become available. The focus of EasyProt is on the programmer and the process of development, simplifying adoption of a principled framework for reproducible experimentation.

Systems research in computer science introduced similar modular systems for various applications. Click [5] used a similar architecture to build configurable

routers. SEDA [6] extended this architecture to general Internet services with an emphasis on horizontal expansion. EasyProt uses a model similar to SEDA in which each element waits on a single queue for messages. To maintain the order of the messages received and to avoid conflict among threads attempting to access shared state, each element in EasyProt runs in its own thread rather than using thread pools as in SEDA.

Cloud computing opens a new way to share data and workflows, through the abstract machine image or whole system snapshot exchange [14]. AMIs are stored in cheap storage, are always available, and can be easily used to develop and maintain multiple versions of software. Sharing of software with AMIs is enormously beneficial as the virtual machines provide complete control over the program execution environment, eliminating issues of portability and dependencies. By releasing EasyProt as an AMI, it is merely necessary for the user to launch a virtual machine from the AMI to run and modify workflows. Multiple virtual machines can be launched to compute workflows in parallel. Virtual machines can be configured to the needs of the workflows, according to desired specifications for computing capability and memory, and, to a lesser extent, network bandwidth. Releasing workflows as EasyProt AMIs addresses the challenges of reproducible computation [15,14], allowing experiments to be readily repeated and verified by others.

2 EasyProt Software Architecture

EasyProt provides a programming model and development environment for computationally demanding experimental workflows in computational biology. A workflow is represented by instances of elements running in parallel across many virtual cores, connected into a directed graph representing data dependencies using a simple graph configuration language. During development of workflows on predefined elements, the elements are connected using the graph configuration language while ensuring that the types of data flowing along the connections satisfy the element specifications, and processed data is retrieved from the cache. When designing new workflows, computational biologists create new elements by implementing a clean interface using an API that hides the parallelism and details of the dataflow. During the process of developing and modifying workflows, the modified AMIs, including all intermediate data, can be stored in stable cloud storage, with no need for transferring large datasets over the network.

2.1 Elements

EasyProt elements each perform a well-defined task. Some of these tasks require accessing external web services to acquire data. Some require running external programs. Others are data processing elements that apply functions to the data before passing it on. Element instances are connected in a directed graph that represents the dependencies among the element instances. Each element instance runs in its own thread, supporting parallelism across multiple virtual cores.

The time required to complete the processing is the time of the longest path from a source to a sink in the directed graph, where the length of a path is given by its running time, with the goal being that this time is limited only by the data dependencies.

An element is represented by an abstract Java class with the core functionality hidden behind a clean interface and shared among all elements. Defining a new element for the user's own needs requires writing two abstract functions in the abstract Java class: an initialization function and a task function. Most of the code necessary for defining a new element is task specific. The element designer is presented with a clean API for retrieving and sending messages through the graph. Instructions for defining new elements are contained on the AMI in well-documented code and examples.

2.2 Graph Configuration Language

Fig. 1 lists a sample graph configuration for a workflow using the elements defined in Section 3. Each configuration file consists of two sections: an initialization section and a connections section. The initialization section defines each instance of the elements that are to be used along with their jobs and initialization strings. The connections section defines the connections among the element instances. For each statement in the connections section the first element instance is connected directly upstream of all other element instances appearing in the statement. Each element defines the types of the elements that can appear directly upstream and directly downstream of it. The elements used in the sample graph configuration are defined in Section 3, and described in more detail, with their jobs and initialization string semantics, on the AMI. The language is implemented as a grammar in ANTLR [16], a Java-based parser generator, and it can be easily extended to accommodate future enhancements.

2.3 Type System

Elements are free to pass custom types along the graph. EasyProt wraps every message in a wrapper type, transparent to the user, before sending it along the graph, providing a uniform interface for messages. A collection of types has been defined, designed for workflows in proteomics, interactomics and comparative genomics. `Proteome` and `Interactome` are generic types that can store proteins and interactions in various formats. A collection of types that can be stored in proteomes and interactomes have been defined, including various protein identifier schemas, annotated proteins, protein amino acid sequences, and protein interactions in various formats. For many elements in the sample workflows described in Section 3, proteomes and interactomes are the units of data passed through the configuration graph; however, some elements combine proteomes or interactomes and release data in a custom output format. Message types have been defined for these custom formats. Designers of new elements are free to use the predefined types or to define new customized types most suitable for their data.

INITIALIZATION

Interactome	int1	“iRefIndex\t9606\tall”	“”
Interactome	int2	“iRefIndex\t7227\tall”	“”
MitabToCache	mtc	“2”	“”
InteractomeFilter	ifil	“2”	“”
InteractomeToCache	itc	“2”	“”
ProteinLinearizer	plin1	“2”	“”
ProteinLinearizer	plin2	“2”	“”
Sequence	seq1	“1”	“”
Sequence	seq2	“1”	“”
SequenceToCache	stc	“2”	“”
Blast	blas	“1.0E-9:inter”	“/easyprot/workspace/blast/blast1/”
BlastToCache	btc	“1”	“”
InteractomeDegree	degr	“2”	“”
Go	go	“2”	“”
MergeAnnotations	merg	“2”	“2”
AnnotationsToCache	atc	“2”	“”
Produles	prod	“P\t5\t20\t1.5\t0.05\t50”	“/easyprot/workspace/produles/”
ModuleAlignmentToCache	matc	“1”	“”
Time	time	“1”	“”
Modularity	modu	“1”	“”
Size	size	“1”	“”
Density	dens	“1”	“”
Overlap	over	“1”	“”
Evolution	evol	“1”	“”
Coverage	cove	“1”	“”
Components	comp	“1”	“”
Summary	summ	“1”	“”
AnalysisTableToCache	attc	“9”	“”
VieprotML	vpml	“1”	“forward”

CONNECTIONS

int1	mtc ifil ;
int2	mtc ifil ;
ifil	itc plin1 plin2 degr prod modu cove ;
plin1	seq1 seq2 ;
plin2	go ;
seq1	stc blas ;
seq2	stc blas ;
blas	btc prod ;
degr	merg ;
go	merg ;
merg	atc vpml ;
prod	matc time modu size dens over evol cove comp vpml ;
time	attc summ vpml ;
modu	attc summ vpml ;
size	attc summ vpml ;
dens	attc summ vpml ;
over	attc summ vpml ;
evol	attc summ vpml ;
cove	attc summ vpml ;
comp	attc summ vpml ;
summ	attc vpml ;

Fig. 1. Graph configuration for a sample workflow

2.4 Cache

EasyProt includes a cache that facilitates reuse of intermediate results. The cache separates the data from the workflows. Each object passed as a message includes a description that records key elements through which the message has travelled. These descriptions are written in string format both as file names with a timestamp and internally. The timestamps allow versioning of the data and the descriptions store the provenance, which allows repeatability. Special elements control reading from the cache and writing to the cache. Caching elements have been defined for all message types currently supported, and it is easy to define new caching elements for custom message types. The cache is organized into subdirectories corresponding to message types. As seen in Fig. 1, experimenters decide which data is cached by calling the appropriate caching elements.

3 Sample Workflows

To demonstrate the usefulness of the EasyProt system, we designed a collection of workflows for comparative interactomics, defining elements for acquiring and annotating protein data and comparing algorithms. A protein interaction network is an undirected graph with nodes that are proteins, and edges that indicate protein interactions between the endpoints. An interactome is a large protein interaction network that ideally includes all protein interactions on an organism's proteome [17]. Comparative interactomics is the comparison of interactomes across organisms. A multiprotein module is a collection of proteins that work together to perform a common task, for example, a protein complex or a signaling pathway [18]. Given two interactomes, G_A and G_B, a module alignment is a set of possibly-overlapping multiprotein modules in G_i for $i \in \{A, B\}$; a one-to-one mapping from the multiprotein modules in G_A to the multiprotein modules in G_B; and a many-to-many mapping between the proteins in aligned modules. The sample workflows acquire current high-quality protein data for multiple organisms, and apply and analyse algorithms to detect multiprotein modules conserved in these organisms.

Several key elements designed for these workflows are defined below.

1. `Interactome`: acquires interactome datasets from various sources.
2. `InteractomeFilter`: removes extraneous data and simplifies the format for later computation.
3. `ProteinLinearizer`: converts interactomes to proteomes.
4. `Sequence`: obtains amino acid sequences for the proteomes from external web services.
5. `Blast`: performs pairwise protein sequence similarity comparisons between proteomes.
6. `InteractomeDegree`: computes the number of interactions for each protein in an interactome.
7. `Go`: obtains protein function annotations.

8. `Produles`: finds multiprotein modules conserved during evolution using the Produles [18] algorithm. Other algorithms are placed in similar elements.
9. `MergeAnnotations`: accepts the output from several elements and produces a unified annotated proteome.
10. `Time`, `Modularity`, `Size`, `Density`, `Overlap`, `Evolution`, `Coverage`, `Components`: compute module alignment evaluation statistics [18].
11. `Summary`: compiles statistics on the module alignment for visualization.
12. `VieprotML`: generates output suitable for visualization of module alignments.

Using workflows similar to Fig. 1, multiple algorithms to detect multiprotein modularity conserved during evolution were compared and evaluated on current datasets [18].

4 Experiments and Results

Versions of the sample workflow in Fig. 1 were applied to the organisms *Homo sapiens*, *Drosophila melanogaster*, and *Saccharomyces cerevisiae* with interactome sizes: *H. sapiens*: 12,952 proteins, 71,035 interactions; *D. melanogaster*: 10,192 proteins, 41,050 interactions; *S. cerevisiae*: 6,083 proteins, 175,113 interactions. Three versions, *H. sapiens vs. D. melanogaster*, *D. melanogaster vs. S. cerevisiae*, and *S. cerevisiae vs. H. sapiens*, were run in parallel on six separate EC2 instances with two configurations of processing capability and memory capacity. Table 1 lists the running times. As the workflows run in parallel, the total time required is the maximum of the individual timings.

Table 1. Time to run sample workflows derived from Fig. 1. The workflows run in parallel so the total time required for all datasets is the maximum of the timings. The EC2 Large instances are applied with 6GB of available RAM and the EC2 High-Memory Quadruple Extra Large instances are applied with 65GB of available RAM.

Dataset	**EC2 Large**	**EC2 High-Memory Quadruple Extra Large**
H. sapiens vs. *D. melanogaster*	7h 32m	7h 18m
D. melanogaster vs. *S. cerevisiae*	7h 22m	7h 9m
S. cerevisiae vs. *H. sapiens*	7h 21m	7h 14m
Complete	7h 32m	7h 18m

Interestingly, the time required for the virtual machines with vastly differing amounts of computing capabilities and memory is quite similar. This seems to be due mainly to the large amount of network requests from web services in these workflows that are limited by the speed of the external servers rather than the computing capability, amount of memory, or network bandwidth of the virtual machines.

The more powerful virtual machines are faster for computationally intensive workflows. To demonstrate this, Table 2 compares the timings on variants of the

sample workflow from Fig. 1 that use data from the cache rather than retrieving data through the network. For these computationally demanding workflows, the more powerful virtual machines require less than three quarters of the time required by the less powerful virtual machines.

Table 2. Time to run sample workflows using data from the cache with the same virtual machines and settings as Table 1

Dataset	**EC2 Large**	**EC2 High-Memory Quadruple Extra Large**
H. sapiens vs. *D. melanogaster*	70m 38s	51m 14s
D. melanogaster vs. *S. cerevisiae*	17m 16s	23m 47s
S. cerevisiae vs. *H. sapiens*	21m 28s	29m 13s
Complete	70m 38s	51m 14s

5 Using EasyProt

EasyProt is distributed as a public AMI that runs on Amazon EC2. The AMI is registered with manifest easyprot-ami/easyprot.img.manifest.xml. Upon launching the AMI, one can type from any directory "easyprot X configFile" to launch EasyProt, where configFile is the name of a configuration file stored in the directory /easyprot/configurations, and X is the amount of memory available to EasyProt in the format nT where n is an integer and $T \in \{m, g\}$, where m specifies megabyte and g specifies gigabyte. Thirty sample workflow configuration files are provided including the sample workflow in Fig. 1. Instructions are also provided in the welcome message on the AMI. Documentation explaining how to use the elements currently defined is provided in the folder /easyprot/documentation and in the source Javadoc. EasyProt, including all source code, is released under the GNU General Public License, Version 3.

The directory structure on the AMI is as follows:

- easyprot
  - cache: contains the EasyProt cache
  - config: contains 30 sample workflow configuration files including Fig. 1
  - documentation: contains documentation for the elements also found in the source Javadoc
  - language: contains the ANTLR grammar and code
  - license: contains open source license
  - project: contains the project with Java source code in subdirectory src
  - scripts: contains the easyprot script that compiles the configuration file, compiles the project, and launches EasyProt
  - workspace: contains temporary workspace and executables for external programs

At the time of this writing, EasyProt supports 61 elements. Documentation for using the elements in the most current version, with allowable connections and specification of jobs and initialization strings, can be found in the file /easyprot/documentation/README on the AMI and in the source Javadoc.

6 Perspectives and Discussion

The power of the EasyProt software architecture is its parallel message-passing design that allows the computation to be distributed across multiple virtual cores and multiple virtual machines. In EasyProt, elements have clearly defined roles and pass messages to other elements allowing for direct horizontal expansion. EasyProt is a programmer-oriented model for reproducible computation in the cloud that supports a parallel element-based approach for using multiple virtual cores and parallel deployment from an AMI for using multiple virtual machines. Enhancements are being implemented for automated partitioning of a single workflow across multiple virtual machines, when desired by the workflow developer, with objectives to minimize message passing across virtual machine boundaries and to group expensive elements with inexpensive elements in order to balance the load.

A clear advantage of the text-based graph configuration language is that it is easy to understand and manipulate directly, allowing for direct modification through a console during the design of ever-changing experiments. For any cost-effective cloud computing programming model, it is essential to minimize transfer of data over the network. The EasyProt model offers immediate availability from any virtual machine running the AMI with no local installation. We are investigating the merits of a hybrid programming environment where flexible console-based access is supplemented with graphical tools that run locally.

MapReduce [19] is a method of parallelization that may increase efficiency for particular highly repetitive tasks. MapReduce elements can be directly implemented in the EasyProt framework. However, preexisting systems such as Hadoop [20] are highly optimized for MapReduce. If a programmer desires to use MapReduce within the EasyProt framework, the recommended option is to design an element that passes data to a MapReduce optimized system.

Comparing the cost of development in the cloud with the cost of development on a high-performance locally managed cluster is not straightforward, as there are hidden costs associated with purchasing and maintaining any cluster. A discussion of cost effectiveness with references to detailed studies, and the conclusion that development in the cloud is cost effective, can be found in [1].

7 Conclusion

We present EasyProt, a parallel software architecture designed to be workflow software for the computational biologist, to bring cloud resources to experimenters who must face the reality of programming increasingly demanding computational workflows on large datasets. We hope that this system will be broadly useful to computational biologists as a cost-effective solution to the challenges of designing and testing time-consuming experimental workflows reproducibly.

Acknowledgments. We thank Nicholas Kong and Richard M. Karp for helpful conversations, and Bonnie Kirkpatrick for careful reading of the manuscript and

discussion. We also thank the anonymous reviewers for their comments that helped to improve the paper. EasyProt is supported in part by NSF Grant CCF-1052553.

References

1. Stein, L.D.: The case for cloud computing in genome informatics. Genome Biology 11(5), 207 (2010)
2. Khalidi, Y.A.: Building a cloud computing platform for new possibilities. Computer 44(3), 29–34 (2011)
3. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., et al.: Examining the challenges of scientific workflows. Computer 40(12), 24–32 (2007)
4. Lord, H.D.: Improving the application development process with modular visualization environments. Computer Graphics 29(2), 10–12 (1995)
5. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, F.: The Click modular router. ACM Trans. on Computer Systems 18(3), 263–297 (2000)
6. Welsh, M., Culler, D., Brewer, E.: SEDA: an architecture for well-conditioned, scalable internet services. In: Proc. of the 18th Symposium on Operating Systems Principles, SOSP 2001 (2001)
7. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., et al.: Above the clouds: a Berkeley view of cloud computing. EECS Department, University of California, Berkeley UCB/EECS-2009-28 (2009)
8. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M. (eds.): Workflows for e-Science: Scientific Workflows for Grids. Springer, Heidelberg (2006)
9. Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13, 219–237 (2005)
10. Juve, G., Deelman, E.: Scientific workflows in the cloud. In: Cafaro, M., Aloisio, G. (eds.) Grids, Clouds and Virtualization, pp. 71–91. Springer, Heidelberg (2010)
11. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., et al.: Taverna: a tool for building and running workflows of services. Nucleic Acids Research 34(Web Server issue), W729–W732 (2006)
12. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., et al.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience 18, 1039–1065 (2006)
13. Linke, B., Giegerich, R., Goesmann, A.: Conveyor: a workflow engine for bioinformatic analyses. Bioinformatics 27(7), 903–911 (2011)
14. Dudley, J.T., Butte, A.J.: In silico research in the era of cloud computing. Nature Biotechnology 28(11), 1181–1185 (2010)
15. Donoho, D.L., Maleki, A., Rahman, I.U., Shahram, M., Stodden, V.: Reproducible research in computational harmonic analysis. Computing in Science and Engineering 11(1), 8–18 (2009)
16. Parr, T.J., Quong, R.W.: ANTLR: a predicated-LL(k) parser generator. Software-Practice and Experience 25(7), 789–810 (1995)
17. Klipp, E., Liebermeister, W., Wierling, C., Kowald, A., Lehrach, H., Herwig, R.: Systems Biology: A Textbook. Wiley-VCH, Weinheim (2009)
18. Hodgkinson, L., Karp, R.M.: Algorithms to detect multiprotein modularity conserved during evolution. IEEE/ACM Trans. on Computational Biology and Bioinformatics (September 27, 2011), IEEE Computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/TCBB.2011.125

19. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
20. Bialecki, A., Cafarella, M., Cutting, D., OMalley, O.: Hadoop: a framework for running applications on large clusters built of commodity hardware, Wiki at, http://lucene.apache.org/hadoop

Bit-Parallel Multiple Pattern Matching

Tuan Tu Tran, Mathieu Giraud, and Jean-Stéphane Varré

LIFL, UMR 8022 CNRS, Université Lille 1, France
INRIA Lille, Villeneuve d'Ascq, France

Abstract. Text matching with errors is a regular task in computational biology. We present an extension of the bit-parallel Wu-Manber algorithm [16] to combine several searches for a pattern into a collection of fixed-length words. We further present an OpenCL parallelization of a redundant index on massively parallel multicore processors, within a framework of searching for similarities with seed-based heuristics. We successfully implemented and ran our algorithms on GPU and multicore CPU. Some speedups obtained are more than 60×.[1]

Keywords: bit parallelism, pattern matching, sequence comparison, neighborhood indexing, GPU, OpenCL.

1 Introduction

With the advances in "Next Generation Sequencing" technologies (NGS), the data to analyze grows even more rapidly than before and requires very efficient algorithms. In *read mapping*, the goal is to map short "reads" produced by NGS on a reference genome; another example is *metagenomics analyses*, where one of the goals can be to identify at which species belongs each read. Such applications require to search a short sequence against a set of sequences. All these problems are thus related to a generic basic problem where we query a *pattern* into one or several *texts*, allowing some *errors*:

Problem 1 (Approximate Pattern Matching in Large Sequences). Given a pattern p, a parameter e and a set of sequences $\mathcal{S}$ over an alphabet Σ, find all occurrences of p in $\mathcal{S}$ within a Levenshtein distance of e.

The Levenshtein distance between two words is the minimal number of insertions, deletions and substitutions needed to transform one word into the other one. In the case of NGS read mapping, the lengths of the patterns are typically from several dozens to several hundreds. The set of sequences can be either a large number of short sequences or a unique large sequence. There are multiple strategies to address this problem (see Section 2.2). The well-known *seed-based heuristics* assume that there is a conserved short word between the pattern and its occurrence in the text. There are numerous tools implementing this strategy,

[1] This research was partially supported by the French ANR project MAPPI. Cards for experiments were provided through a "Action Incitative LIFL" grant.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 292–301, 2012.

including the popular BLAST [1], but also tools using improved seeds such as PatternHunter [9] or YASS [12]. Such methods involve several stages, including a neighborhood extension that can be formalized as follows:

Problem 2 (Approximate Pattern Matching in Fixed-Length Word). Given a pattern p and a parameter e, find all words in a set of fixed-length words such that the Levenshtein distance with p is at most e.

The length of such fixed-length words is often small (4, 8, 16 in this paper), and corresponds only to the neighborhood of matched seeds. Positive results at this step lead to alignments on the full pattern length to solve Problem 1.

GPU/manycore Computing. Graphics processing units (GPUs) were used in bioinformatics since 2005 [4]. Since then, lots of different studies were proposed on Smith-Waterman sequence comparisons or other bioinformatics applications (review in [15]). Computing with GPUs was firstly done by tweaking graphics primitives. The CUDA libraries, released by NVIDIA in 2007[2], and the OpenCL standard[3] now enable easy programming on GPU/manycore architectures. The same OpenCL code can be compiled and optimized for different platforms [5]. As of today, at least five different implementations of this standard are available: NVIDIA, AMD, Apple, IBM, and Intel. In the following years, the OpenCL standard could become a practical standard for parallel programming.

Contents. This paper brings two contributions. In Section 3, we propose a simple extension to the bit-parallel algorithms of [16] to a collection of fixed-length words to address Problem 2. In Section 4, extending ideas from [13], we show how, for solving Problem 1 with a seed-based heuristic, a redundant neighborhood indexing is more efficient than an offset indexing and scales well on GPU/manycore architectures. Performance tests on CPU and GPU are given in Section 5.

It should be noted that Hirashima *et al.* also studied bit-parallel algorithms on CUDA [8], but their algorithms were optimized for strings over a binary alphabet for stringology studies and do not apply here.

2 Background

We provide here some background in bit-parallelism (Section 2.1), as well as in seed-based heuristics and neighborhood indexing (Section 2.2).

2.1 Bit Parallel Matching

Text searching using bit-parallelism emerged in the early 90's. The approach consists in taking advantage of the parallelism of bit operations, encoding the states of a matching automaton into a machine word seen as a bit array. Ideally,

[2] `http://www.nvidia.com/cuda`
[3] `http://www.khronos.org/opencl`

these algorithms divide the complexity by w, where w is the length of a machine word. The book [11] is an excellent reference on the subject.

The Shift-or algorithm for exact pattern matching [2] is one of the first algorithms using this paradigm. In 1992, Wu and Manber [16] proposed an approximate matching algorithm. The Wu-Manber algorithm (called BPR, for Bit-Parallelism Row-wise, in [11]) allows substitution, insertion and deletion errors, and was implemented in the `agrep` software.

Exact Matching. The pattern p of length m is encoded over a bit array R of length m. Characters of the text t are processed one by one, and we denote by $R^{[j]}$ the value of R once the first j letters of the text have been read. More precisely, the i^{th} bit of $R^{[j]}$ equals 1 if and only if the first i characters of the pattern $(p_1...p_i)$ match exactly the last i characters of the text $(t_{j-i-1}...t_j)$. The first bit of $R^{[j]}$ is thus just the result of the matching of $(p_1 = t_j)$, and, when $i \geq 2$, the i^{th} bit $R^{[j]}(i)$ of $R^{[j]}$ is obtained by:

$$R^{[j]}(i) = \begin{cases} 1 \text{ if } R^{[j-1]}(i-1) = 1 \text{ and } (p_i = t_j) \text{ (match)} \\ 0 \text{ otherwise} \end{cases}$$

With bitwise operators and (&) and shift ($\ll$), this results in Algorithm 1.

Algorithm 1. Exact Bit-Parallel Matching

$$\begin{cases} R^{[0]} \leftarrow 0^m \\ R^{[j]} \leftarrow \left((R^{[j-1]} \ll 1) \mid 0^{m-1}1 \right) \ \& \ B[t_j] \end{cases}$$

The *pattern bitmask* B is a table with $|\Sigma|$ bit arrays constructed from the pattern, such that $B[t_j](i) = 1$ if and only if $(p_i = t_j)$. This algorithm works as long as $m \leq w$, where w is the length of the machine word, and needs $O(z)$ operations to compute all $R^{[j]}$ values, where z is the length of the text.

Approximate Matching. To generalize the matching up to e errors, we now consider $e+1$ different bit arrays $R_0, R_1, ...R_e$, each one of length m. The i^{th} bit of $R_k^{[j]}$ equals 1 if and only if the first i characters of the pattern match a subword of the text finishing at t_j with at most k errors, leading to Algorithm 2. Due to insertion and deletion errors, the length of a match in the text is now in the interval $[m-e, m+e]$.

This algorithm works as long as $m + e \leq w$, but now takes $O(ez)$ time. BPR has been reported as the best unfiltered algorithm in DNA sequences, for low error levels and short patterns (p. 182 of [11]). We thus focused on this algorithm instead of theoretically better ones such as BNDM, implemented in the `nrgrep` software [10]. Moreover, BPR is more regular than other solutions, enables a better performance on processors with large memory words such as those with SIMD instructions.

Algorithm 2. BPR Matching

$$\begin{cases} R_k^{[0]} \leftarrow 0^{m-k}1^k \\ R_0^{[j]} \leftarrow \left((R_0^{[j-1]} \ll 1) \mid 0^{m-1}1\right) \ \& \ B[t_j] \quad \text{(match)} \\ R_k^{[j]} \leftarrow \underbrace{\left((R_k^{[j-1]} \ll 1) \ \& \ B[t_j]\right)}_{\text{(match)}} \mid \underbrace{R_{k-1}^{[j-1]}}_{\text{(insertion)}} \mid \underbrace{(R_{k-1}^{[j-1]} \ll 1)}_{\text{(substitution)}} \mid \underbrace{(R_{k-1}^{[j]} \ll 1)}_{\text{(deletion)}} \mid \underbrace{0^{m-k}1^k}_{\text{(init)}} \end{cases}$$

2.2 Seed-Based Heuristics and Redundant Neighborhood Index

Preprocessing and Indexes. A common way to speed-up pattern matchings is to preprocess the text, for example by building an index in which performing a query of a pattern can be achieved in constant or linear time. A lot of work has been done to build non-redundant indexes in order to save memory, such as suffix trees or suffix arrays [6]. Obviously, there is always a trade-off between memory efficiency and time efficiency. Thus, simpler index techniques, even with some redundancy, could achieve better time performance.

Seed-Based Indexing. In seed-based indexing, the pattern p is divided in two parts: a *seed* p_s of size W, which has to occur exactly, and a *neighborhood* p_n of length ℓ. In the *filtering phase*, we search for occurrences of the seed in the index and retrieve the list of all its neighborhoods. In the *finishing phase*, the neighborhood of the pattern is compared to the neighborhoods of the occurrences of the seed.

Complete seed-based heuristics can include further finishing stages. Moreover, designing efficient seeds is a wide area of research, including spaced seeds and their extensions (see [3] for a review). We will not address these problems here, but focus on the *storage of neighborhoods*: how can we, for each seed, have access to the list of all its occurrences in the index ? We now follow the discussion of [14] to present different ways to store these neighborhoods.

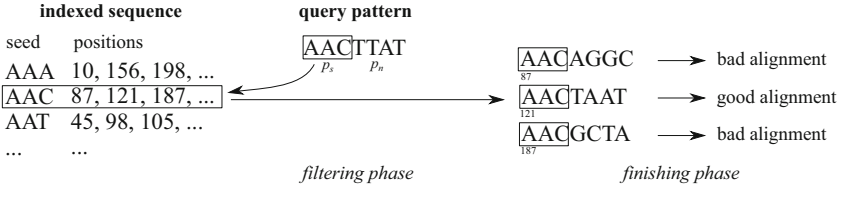


Fig. 1. Offset indexing [14]. During the filtering phase, the seed p_s (here `AAC`) is used to access to the index. Then the finishing phase get neighborhoods of this seed (one memory access per position), and compare them against p_n. If N is the total number of neighborhoods, each offset takes $\log N$ bits, thus index size is $N \times \log N$ bits.

Offset Indexing. In the usual *offset indexing* approach, depicted on Figure 1, an offset is stored for each seed position. For each query position, each hit returned by the filtering phase leads to an iteration of the finishing phase. This iteration accesses some neighborhoods of the positions. These memory accesses are *random*, that are unpredictable and non contiguous. Such accesses are not efficiently cached and require high latencies [7]. This is still true for GPUs: despite high internal memory bandwidths, random access patterns decrease efficiency.

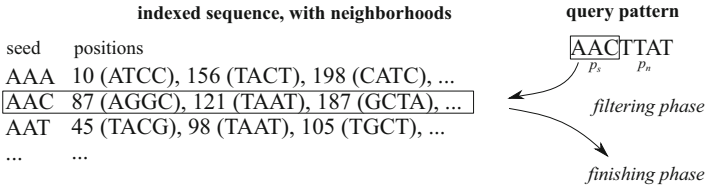


Fig. 2. Neighborhood indexing [14]. All neighborhoods are stored along the offset of each seed occurrence. One unique memory access gives all the data needed by the finishing phase. For nucleotide characters, stored in 2 bits, the overall index size is then equal to $N \times (\log N + \ell)$ bits, where ℓ is the length of the neighborhood.

Neighborhood Indexing. A way to reduce the computation time is thus to avoid as far as possible such random memory accesses. In [13], a *neighborhood indexing* approach has been proposed. The idea is to directly store in the index the neighborhood of size ℓ for every seed occurrence (Figure 2). Thus all neighborhoods corresponding to a seed are obtained through a single contiguous memory access.

This index is redundant, as every character of the text will be stored in the neighborhoods of ℓ different seeds: the neighborhood indexing enlarges the size of the index. However, it can improve the computation time by reducing the random memory access. In [13], the authors claimed that the neighborhood indexing speeded up the execution time by a factor ranging between 1.5 and 2 over the offset indexing.

3 Multiple Fixed-Length Bit-Parallel Matchings

In this section, we propose an extension of BPR (Algorithm 2, [16]) which solves Problem 2 and takes care of memory accesses. More formally, we compare a pattern p_n of length m against a collection of words t^1, t^2, ... t^n of length $\ell = m + e$, allowing at most e errors (substitutions, insertions, deletions).

The existing bit-parallel algorithms for multiple pattern matching (review in Section 6.6 of [11]) match a set of patterns within a large text. Our setup is different, as we want to match one pattern with several texts. Of course, one could reverse the multiple pattern matching algorithms and build an automaton on a set of all neighborhoods. This would result in a huge automaton, and the algorithm would not be easily parallelizable.

t^1 = ATCG, t^2 = GGAC

t^3 = AGCG, t^4 = AGTC

$\widehat{B}$[AGAA] | 0 0 1 | 0 0 0 | 0 0 1 | 0 0 1 | $\longrightarrow$

$\widehat{B}$[TGGG] | 0 1 0 | 0 0 0 | 0 0 0 | 0 0 0 | $\longrightarrow$

$\widehat{B}$[CACT] | 1 0 0 | 0 0 1 | 1 0 0 | 0 1 0 | $\longrightarrow$

$\widehat{B}$[GCGC] | 0 0 0 | 1 0 0 | 0 0 0 | 1 0 0 | $\longrightarrow$

	t^1	t^2	t^3	t^4
$R_0^{[0]}$	0 0 0	0 0 0	0 0 0	0 0 0
$R_1^{[0]}$	0 0 1	0 0 1	0 0 1	0 0 1
$R_0^{[1]}$	0 0 1	0 0 0	0 0 1	0 0 1
$R_1^{[1]}$	0 1 1	0 0 1	0 1 1	0 1 1
$R_0^{[2]}$	0 1 0	0 0 0	0 0 0	0 0 0
$R_1^{[2]}$	***1*** 1 1	0 0 0	0 1 1	0 1 1
$R_0^{[3]}$	***1*** 0 0	0 0 1	0 0 0	0 0 0
$R_1^{[3]}$	***1*** 1 0	0 1 0	***1*** 0 0	0 1 0
$R_0^{[4]}$	0 0 0	0 0 0	0 0 0	0 0 0
$R_1^{[4]}$	***1*** 0 0	***1*** 1 1	0 0 0	***1*** 0 0

Fig. 3. Execution of the algorithm 3 on 12-bit machine words. The pattern p = ATC, of length $m = 3$, is compared against $n = 4$ words with up to $e = 1$ error. The text data t ={ATCG, GGAC, AGCG, AGTC} is stored in a stripped layout as AGAA TGGG CACT GCGC. After 2 iterations, there is one approximate match for t^1 (AT, one insertion). After 3 iterations, there is one exact match for t^1, and one approximate match for t^3 (AGC, one substitution). After 4 iterations, there are three approximate matches, for t^1 (ATCG, one deletion), t^2 (AC, one insertion) and t^4 (AGTC, one deletion).

Algorithm. The idea of our algorithm is to store n fixed-length words into a machine word, so n matchings can be done simultaneously. As in BPR, to have a matching up to e errors, we consider $e+1$ different bit arrays $R_0, R_1, ...R_e$, but each one is now of size mn, that is n slices of m bits. If $1 \leq r \leq n$ and $1 \leq i \leq m$, the i^{th} bit of the r^{th} slice of $R_p^{[j]}$ equals 1 if and only if the first i characters of the pattern match the last i characters of the r^{th} text $(t^r_{j-i-1}...t^r_j)$ with at most k errors. We thus obtain Algorithm 3.

Algorithm 3. Multiple Fixed-Length BPR (mflBPR) Matching

$$\begin{cases} R_k^{[0]} \leftarrow (0^{m-k}1^k)^n \\ R_0^{[j]} \leftarrow \Big((R_0^{[j-1]} \ll 1) \mid (0^{m-1}1)^n \Big) \ \& \ \widehat{B}[\widehat{t_j}] \quad \text{(match)} \\ R_k^{[j]} \leftarrow \underbrace{\Big((R_k^{[j-1]} \ll 1) \ \& \ \widehat{B}[\widehat{t_j}] \Big)}_{\text{(match)}} \mid \underbrace{R_{k-1}^{[j-1]}}_{\text{(insertion)}} \mid \underbrace{(R_{k-1}^{[j-1]} \ll 1)}_{\text{(substitution)}} \mid \underbrace{(R_{k-1}^{[\mathbf{j}]} \ll 1)}_{\text{(deletion)}} \mid \underbrace{(0^{m-k}1^k)^n}_{\text{(init)}} \end{cases}$$

Figure 3 shows a run of this algorithm. Compared to BPR, the initialization is $(0^{m-k}1^k)^n$ instead of $0^{m-k}1^k$. This initialization puts 1's at the k first bits of each slice, thus overriding any data shifted from another slice. Moreover, to allow better memory efficiency:

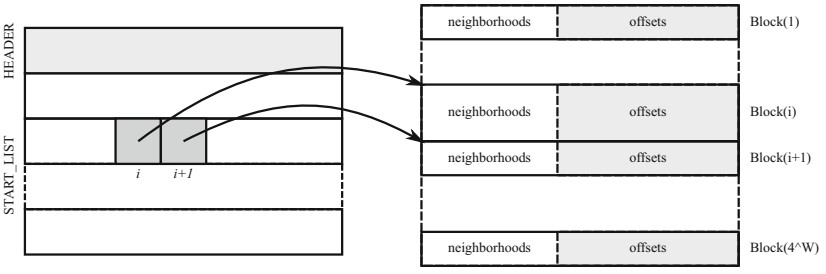


Fig. 4. Structure of the index file. The **START_LIST** contains the positions of 4^W Blocks. The main part is the list of Blocks, each one containing, for a given seed, its neighborhoods as well as its offsets.

- The set of n fixed-length words $t = \{t^1, t^2, ..., t^n\}$ is stored and accessed through a stripped layout, as each access j returns the j^{th} characters of every word: $\widehat{t_j} = t_j^1 t_j^2 ... t_j^n$
- The block mask $\widehat{B}[t_j^1 t_j^2 ... t_j^n] = B[t_j^1]B[t_j^2]...B[t_j^n]$ is thus now larger, having $|\Sigma|^n$ bit arrays (instead of $|\Sigma|$). As in BPR, the computation of this table still depends only of the pattern. This table is somehow redundant, but now allows the match of n characters with one unique memory access.

This algorithm works as long as $mn \leq w$, where w is the length of the machine word, and needs $O(ez/n)$ operations to compute all $R_k^{[j]}$ values, where z is the total length of all texts. Comparing to the BPR algorithm, there are n times less operations. Of course, the limiting factor is again the size of the machine word.

4 Redundant Parallel Neighborhood Indexing

We now describe our redundant neighborhood index for DNA ($|\Sigma| = 4$), and show how it may be used on regular processors as well as on GPUs. The structure of the neighborhood index file is depicted on Figure 4. Searching for a pattern $p = p_s p_n$ is done through the following natural steps:

Pattern pre-processing
Compute the pattern bitmask $B(p_n)$ (for BPR) or $\widehat{B}(p_n)$ (for mflBPR)
Filtering phase
Retrieve the position of Block(p_s) in the index
Finishing phase
Compare p_n against all the neighborhoods of Block(p_s)

This index works either with BPR or mflBPR, or with any other neighbor comparing method. In all cases, there are very few random memory accesses since the Block(p_s) is stored contiguously in the memory.

Usage with OpenCL Devices. All the index data are precomputed and transferred only once to the device. Then the application runs looping on each query. The pattern pre-processing as well as the $\mathsf{Block}(p_s)$ position retrieving are done on the host. Then the block bitmask $B(p_n)$ or $\widehat{B}(p_n)$ and the positions of $\mathsf{Block}(p_s)$ are sent to the global memory of the device. The device is devoted to the finishing phase. Depending on the size of $\mathsf{Block}(p_s)$, several comparing cycles may be run. In each comparing cycle, neighborhoods are distributed into different work groups and loaded in the local memory of each work group, and processed by several work items. The positions of the matching neighborhoods are then written back to a result array in the global memory, then transferred back to the host. This index is intrinsically parallel, as the neighborhoods are processed independently.

Index Size. An obvious drawback of the neighborhood indexing is the additional memory requiring to store neighborhoods. The ratio between the overall index sizes of the neighborhood indexing and the offset indexing is $r_\ell = 1 + 2\ell/\log N$. For alignment purposes, considering offsets of $\log N = 32$ bits gives ratios that are acceptable, $r_8 = 1.5$ and $r_{16} = 2$. For example, a 100 Mbp sequence with a neighborhood of size $L = 8$ with small seeds ($W \leq 6$) gives an index of size 630 MB, fitting in the main host memory as well as in GPU cards.

5 Performance Results and Perspectives

Testing environment. We benchmarked the algorithms BPR and mpfBPR on GPU and on multicore CPU. The same OpenCL code was used, but with different OpenCL libraries. We thus target these two platforms:

- **GPU:** NVIDIA 480 (30×16 cores, 1.4 GHz, 1.5 GB RAM), with the OpenCL library was NVIDIA GPU Computing SDK 1.1 beta.
- **CPU:** Intel Xeon E5520 (8 cores, 2.27 GHz, 8 MB cache), with the OpenCL library was AMD APP SDK 2.4.

We also tested a C++ "CPU serial" version, which ran on only one core of the CPU. Programs were compiled by `GNU g++` with the `-O3` option. The host computer had 8 GB RAM.

Methodology. Tests were run on the first 100 Mbp of the human chromosome 1. We measured the performance of the algorithms in *millions of words matched per second* (Mw/s). This normalization allowed to benchmark the problem 2 independently. For the problem 1, the number of words was the total number of neighborhoods, removing the bias due to seed selection. We ran searches on 10 successive patterns, but we saw no significant difference between 1, 10, or 100 patterns, as soon as enough computations hided the transfer times.

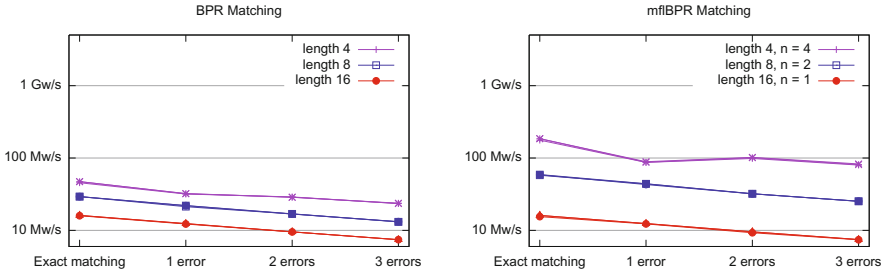


Fig. 5. Performance of BPR (left) or mflBPR (right), both on CPU serial version

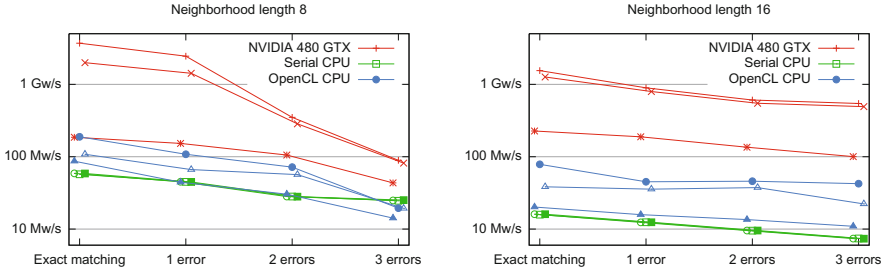


Fig. 6. Performance of the neighborhood indexing with a neighborhood length of 8 (left) and 16 (right). For each platform, there are three different curves, corresponding to different seed lengths (3, 4 and 6), hence to different total numbers of neighborhoods. Times used in the OpenCL versions include transfer times between host and device for the queries and the result, but not for the index.

Performance of mflBPR (CPU). Figure 5 shows performance of mflBPR on a CPU. With 32-bit integers, performance gain compared to BPR ranges from 2.73× to 3.92× for words of length 4 and from 1.89× to 2.06× for words of length 8, close to the 4× and 2× theoretical gains.

Performance of redundant neighborhood index (CPU and GPU). Figure 6 shows performance of the whole index. In the most simple instance (neighborhoods of size 8, no error), the serial CPU implementation peaks at 59 Mw/s, the OpenCL CPU at 189 Mw/s, and the OpenCL GPU at 3693 Mw/s. In this case, using OpenCL brings speed-ups of about 3.2× on CPU and about 62× on GPU. In the same setup, the offset indexing peaks at 4.0 Mw/s on serial CPU, 108 Mw/s on OpenCL CPU and 1706 Mw/s on OpenCL GPU (data not shown).

When the error rises, performance degrades in both implementations. On small neighborhoods, starting from $e = 2$, the performance of both GPU and CPU versions are limited by the number of matches in the output. However, even in the worst case (7.5 Mw/s, CPU serial implementation, 3 errors, seed size 4 and neighborhood length of size 16), using the neighborhood indexing takes less than 0.06 s for parsing a chromosome with 100 Mbp, while non-indexed approaches using bit parallelism takes 0.9 s with `agrep` and 0.7 s with `nrgrep`.

We thus demonstrated the efficiency of using OpenCL and GPUs to speed-up the neighborhood phase extension in seed-based heuristics.

Perspectives. Further work could include a complete evaluation of the redundant index, including a study on the influence of the seed design. It should be noted that our OpenCL code already works both on NVIDIA and AMD SDKs. However, tests on ATI GPU cards (Radeon 5870) give now poor performance (best result peaking at 39.6 Mw/s on a smaller index, not shown). We would thus like to benchmark and optimize our code on other OpenCL platforms.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. 215(3), 403–413 (1990)
2. Baeza-Yates, R.A., Gonnet, G.H.: A new approach to text searching. SIGIR Forum 23(3-4), 7 (1989)
3. Brown, D.G.: A survey of seeding for sequence alignment. In: Bioinformatics Algorithms: Techniques and Applications, pp. 126–152 (2008)
4. Charalambous, M., Trancoso, P., Stamatakis, A.P.: Initial Experiences Porting a Bioinformatics Application to a Graphics Processor. In: Bozanis, P., Houstis, E.N. (eds.) PCI 2005. LNCS, vol. 3746, pp. 415–425. Springer, Heidelberg (2005)
5. Gummaraju, J., Morichetti, L., Houston, M., Sander, B., Gaster, B.R., Zheng, B.: Twin peaks: a software platform for heterogeneous computing on general-purpose and graphics processors. In: Parallel Architectures and Compilation Techniques (PACT 2010), pp. 205–216 (2010)
6. Gusfield, D.: Algorithms on Strings, Trees, and Sequences (1997)
7. Hennessy, J.L., Patterson, D.A.: Computer Architecture, A Quantitative Approach. Morgan Kaufmann (2006)
8. Hirashima, K., Bannai, H., Matsubara, W., Ishino, A., Shinohara, A.: Bit-parallel algorithms for computing all the runs in a string. In: Prague Stringology Conference, PSC 2009 (2009)
9. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics 18(3), 440–445 (2002)
10. Navarro, G.: NR-grep: a fast and flexible pattern-matching tool. Software: Practice and Experience 31(13), 1265–1312 (2001)
11. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences (2002)
12. Noé, L., Kucherov, G.: YASS: enhancing the sensitivity of DNA similarity search. Nucleic Acids Research 33(S2), W540–W543 (2005)
13. Peterlongo, P., Noé, L., Lavenier, D., Nguyen, V.H., Kucherov, G., Giraud, M.: Optimal neighborhood indexing for protein similarity search. BMC Bioinformatics 9(534) (2008)
14. Pisanti, N., Giraud, M., Peterlongo, P.: Filters and Seeds Approaches for Fast Homology Searches in Large Datasets. Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications (2011)
15. Varré, J.-S., Schmidt, B., Janot, S., Giraud, M.: Manycore High-Performance Computing in Bioinformatics. In: Advances in Genomic Sequence Analysis and Pattern Discovery. World Scientific (2011)
16. Wu, S., Manber, U.: Fast Text Searching Allowing Errors. Communications of the ACM 35(10), 83–91 (1992)

A Parallel Space-Time Finite Difference Solver for Periodic Solutions of the Shallow-Water Equation

Peter Arbenz[1], Andreas Hiltebrand[2], and Dominik Obrist[3]

[1] ETH Zürich, Chair of Computational Science, Zürich, Switzerland
arbenz@inf.ethz.ch
[2] ETH Zürich, Seminar for Applied Mathematics, Zürich, Switzerland
[3] ETH Zürich, Institute of Fluid Dynamics, Zürich, Switzerland

Abstract. We investigate parallel algorithms for the solution of the shallow-water equation in a space-time framework. For periodic solutions, the discretized problem can be written as a large cyclic non-linear system of equations. This system of equations is solved with a Newton iteration which uses two levels of preconditioned GMRES solvers. The parallel performance of this algorithm is illustrated on a number of numerical experiments.

1 Introduction

In this paper we consider the shallow water equation as a model for the behavior of a fluid in a rectangular basin $\Omega = (0, L_x) \times (0, L_y)$ which is excited periodically. The excitation is caused by periodic swayings of the ground of the basin with a frequency ω, imposing a periodic behavior of the fluid with the period $T = 2\pi/\omega$ [4].

In the classical approach to solve such problems, the transient behavior of the fluid is simulated starting from an arbitrary initial state. This simulation is continued until some periodic steady-state evolves.

We model the fluid in space-time $\Omega \times [0, T)$. We will impose periodic boundary conditions in time. The discretization of the shallow water equations by finite differences in space *and* time leads to a very large nonlinear system of equations that requires parallel solution. The parallelization is done by domain decomposition where the subdomains partition space *and* time in a natural way. This is a large advantage over the classical approach where only space can be partitioned. At the same time, the number of degrees of freedom is larger by $\mathcal{O}(T/\Delta t)$.

Approaches that admit parallelization in time exist but are quite recent and not very popular yet. In the 'parareal' approach [6], the time interval $[0, T]$ is divided in subintervals. On each of these the given system of ODEs is solved. The global solution is obtained by enforcing continuity at the interfaces. Stoll and Wathen [10] discuss an 'all-at-once' approach to solve a PDE-constrained optimization problem. In this problem a state has to be controlled in such a way that it is driven into a desired final state at time T. The discretization is by finite elements in space and finite differences (backward Euler) in time. A symmetric saddle-point problem has to be solved in this approach.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 302–312, 2012.

2 Governing Equations

The shallow water equations with fringe forcing and Laplacian damping are given by [4,12]

$$\partial_t a(\mathbf{x},t) + (\boldsymbol{u}\cdot\boldsymbol{\nabla})a = -a\,\boldsymbol{\nabla}\cdot\boldsymbol{u}, \qquad \mathbf{x}\in\Omega, \quad t>0, \tag{1a}$$

$$\partial_t \boldsymbol{u}(\boldsymbol{x},t) + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u} = -g\,\boldsymbol{\nabla}h + \varepsilon\Delta\boldsymbol{u} - \Lambda\,\boldsymbol{u}, \qquad \mathbf{x}\in\Omega, \quad t>0, \tag{1b}$$

$$\boldsymbol{u}\cdot\boldsymbol{n} = 0, \qquad \mathbf{x}\in\partial\Omega, \ t>0. \tag{1c}$$

Equations (1a) and (1b) model conservation of mass and momentum, respectively. Here, $\boldsymbol{u}$ is the velocity vector, h is the fluid surface level, z is the ground level, and a is the depth of the fluid (Fig. 1),

$$h(\boldsymbol{x},t) = z(\boldsymbol{x},t) + a(\boldsymbol{x},t). \tag{2}$$

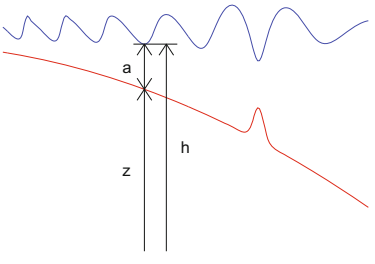


Fig. 1. Illustration of the definition of the ground level z, the water depth a and the surface level h (blue: the fluid surface h; red: the ground surface z)

The ground level z consists of a static part z_0 and a forcing term f oscillating with period T,

$$z(\boldsymbol{x},t) = z_0(\boldsymbol{x}) + f(\boldsymbol{x},t), \qquad f(\boldsymbol{x},t) = f(\boldsymbol{x},t+T).$$

The gravitational constant g in (1b) determines the phase velocity of the shallow water waves as $\sqrt{ga}$. The fringe forcing $-\Lambda\,\boldsymbol{u}$ is used to damp the waves along the boundary $\partial\Omega$ of the computational domain such that it acts like a non-reflecting outflow boundary condition. $\Lambda(\boldsymbol{x})$ is zero away from the boundary such that the forcing term has no effect in most of the interior of Ω. As $\boldsymbol{x}$ approaches $\partial\Omega$, the fringe function Λ rises smoothly to a large value such that the forcing term dominates the other terms in (1b) and we effectively solve the equation $\partial_t\boldsymbol{u} \approx -\Lambda\boldsymbol{u}$ which forces $\boldsymbol{u}$ rapidly toward zero.

The periodicity of the forcing term $f(\boldsymbol{x},t)$ transfers to the solution

$$a(\boldsymbol{x},t) = a(\boldsymbol{x},t+T), \quad \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{u}(\boldsymbol{x},t+T), \qquad \boldsymbol{x}\in\Omega, \ t>0. \tag{3}$$

These equations give rise to periodic boundary conditions in space and time for our model (1) that we solve in $\Omega \times [0, T)$. Using the periodic boundary conditions (3) together with (1c) we get from the conservation of mass (1a) that

$$0 = \int_\Omega \partial_t a \,\mathrm{d}\boldsymbol{x} + \int_\Omega \boldsymbol{\nabla}\cdot(a\boldsymbol{u})\mathrm{d}\boldsymbol{x} = \mathrm{d}_t \int_\Omega a \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} a\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}S = \mathrm{d}_t \int_\Omega a \,\mathrm{d}\boldsymbol{x}, \quad (4)$$

which means that the amount of fluid is conserved over time. This amount is determined by the initial data.

3 Newton Iteration

Equations (1a)–(1b) can be writen in matrix form as

$$\mathcal{A}(a, \boldsymbol{u}) \begin{bmatrix} a \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} 0 \\ -g\boldsymbol{\nabla} z \end{bmatrix} \quad (5)$$

where

$$\mathcal{A}(a, \boldsymbol{u}) = \begin{bmatrix} \partial t + \boldsymbol{u}\cdot\boldsymbol{\nabla} + (\boldsymbol{\nabla}\cdot\boldsymbol{u})\cdot & 0 \\ g\boldsymbol{\nabla} & \partial t + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\cdot - \varepsilon\Delta + \Lambda\cdot \end{bmatrix}. \quad (6)$$

We use Newton's method to solve this nonlinear equation. To that end we linearize (5) at $(a_\ell, \boldsymbol{u}_\ell)$. Substituting $a = a_\ell + \delta a$ and $\boldsymbol{u} = \boldsymbol{u}_\ell + \delta\boldsymbol{u}$ in (5) and omitting higher order terms we obtain

$$\begin{aligned} &\partial_t(\delta a) + \boldsymbol{\nabla}\delta a\cdot\boldsymbol{u}_\ell + \delta a\,\boldsymbol{\nabla}\cdot\boldsymbol{u}_\ell + \boldsymbol{\nabla}a_\ell\cdot\delta\boldsymbol{u} + a_\ell\,\boldsymbol{\nabla}\cdot\delta\boldsymbol{u} = -\partial_t a_\ell - \boldsymbol{\nabla}(a_\ell\boldsymbol{u}_\ell),\\ &\partial_t(\delta\boldsymbol{u}) + (\delta\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u}_\ell + (\boldsymbol{u}_\ell\cdot\boldsymbol{\nabla})\delta\boldsymbol{u} + g\boldsymbol{\nabla}\delta a + \Lambda\delta\boldsymbol{u} - \varepsilon\Delta\delta\boldsymbol{u}\\ &\qquad = -\partial_t\boldsymbol{u}_\ell - (\boldsymbol{u}_\ell\cdot\boldsymbol{\nabla})\boldsymbol{u}_\ell - g\boldsymbol{\nabla}h - \Lambda\boldsymbol{u}_\ell + \varepsilon\Delta\boldsymbol{u}_\ell, \end{aligned}$$

which can formally be written as

$$\mathcal{H}(a_\ell, \boldsymbol{u}_\ell) \begin{bmatrix} \delta a \\ \delta\boldsymbol{u} \end{bmatrix} = \begin{bmatrix} r_h \\ \boldsymbol{r}_u \end{bmatrix} \quad (7)$$

with

$$\mathcal{H}(a_\ell, \boldsymbol{u}_\ell) = \begin{bmatrix} \partial_t + \boldsymbol{u}_\ell\cdot\boldsymbol{\nabla} + \boldsymbol{\nabla}\cdot\boldsymbol{u}_\ell I & (\boldsymbol{\nabla}a_\ell)\cdot + a_\ell\,\boldsymbol{\nabla}\cdot \\ g\boldsymbol{\nabla} & \partial_t + \dfrac{\partial\boldsymbol{u}}{\partial\boldsymbol{x}} + (\boldsymbol{u}_\ell\cdot\boldsymbol{\nabla}) + \Lambda - \varepsilon\Delta \end{bmatrix}.$$

Here, $\partial\boldsymbol{u}/\partial\boldsymbol{x}$ denotes the Jacobian.

A formal algorithm for solving the nonlinear system of equations (5) now reads.

Algorithm 3.1. Newton iteration for periodic solutions of the shallow water equation

1: Choose initial approximations $(a^{(0)}, \boldsymbol{u}^{(0)})$.
2: **for** $k = 0, \ldots,$ `maxIt` -1 **do**
3: Determine residuals according to (5)

$$\begin{bmatrix} r_h^{(\ell)} \\ \boldsymbol{r}_u^{(\ell)} \end{bmatrix} = \begin{bmatrix} 0 \\ -g\boldsymbol{\nabla} z \end{bmatrix} - \mathcal{A}(a^{(\ell)}, \boldsymbol{u}^{(\ell)}) \begin{bmatrix} a^{(\ell)} \\ \boldsymbol{u}^{(\ell)} \end{bmatrix}.$$

4: **If** $\|r_h^{(\ell)}\|^2 + \|\boldsymbol{r}_u^{(\ell)}\|^2 < \eta^2(\|a^{(\ell)}\|^2 + \|\boldsymbol{u}^{(\ell)}\|^2)$ **then** exit.
5: Solve

$$\mathcal{H}(a^{(\ell)}, \boldsymbol{u}^{(\ell)}) \begin{bmatrix} \delta a \\ \delta \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} r_h^{(\ell)} \\ \boldsymbol{r}_u^{(\ell)} \end{bmatrix}$$

for the Newton corrections δa, $\delta \boldsymbol{u}$.
6: Update the approximations $a^{(k+1)} := a^{(\ell)} + \delta a, \quad \boldsymbol{u}^{(k+1)} := \boldsymbol{u}^{(\ell)} + \delta \boldsymbol{u}$.
7: **end for**

4 Discretization

We approximate the shallow-water equation (1) by finite differences in space and time [3,5]. To that end we define grid points

$$(x_i, y_j, t_k) = (i\Delta x, j\Delta y, k\Delta t), \quad \Delta x = L_x/N_x, \ \Delta y = L_y/N_y, \ \Delta t = T/N_t. \quad (8)$$

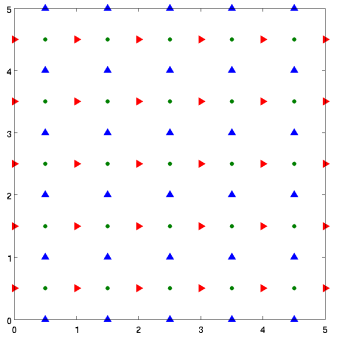


Fig. 2. Grid points for u (▶), v (▲), a (●). Here, $N_x = N_y = 5$.

In this and the following sections we write $\boldsymbol{x} = (x, y)$ and $\boldsymbol{u} = (u, v)$. We approximate the functions a, u, and v at points with (partly) non-integer indices as indicated in Fig. 2,

$$\begin{aligned} a^{(k)}_{i+\frac{1}{2},j+\frac{1}{2}} &\approx a(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}, t_k), \\ u^{(k)}_{i,j+\frac{1}{2}} &\approx u(x_i, y_{j+\frac{1}{2}}, t_k), \\ v^{(k)}_{i+\frac{1}{2},j} &\approx v(x_{i+\frac{1}{2}}, y_j, t_k). \end{aligned}$$

The corresponding grids are called h-grid, u-grid, and v-grid, with N_xN_y, $(N_x-1)N_y$, and $N_x(N_y-1)$ *interior* grid points, respectively. Notice that values of u and v at grid points on the boundary vanish according to (1c).

The finite difference equations corresponding to (1) are defined in one of these grids. Function values of h, u, and v that are required on another grid than where they are defined are obtained through *linear interpolation*. In (1) derivatives in x- and y-direction are replaced by central differences. The derivatives in time are discretized by a fourth-order 'slightly backward-facing' finite difference stencil,

$$\frac{\partial}{\partial t} f(t) \approx \frac{1}{12\Delta t} \left[3f(t+\Delta t) + 10f(t) - 18f(t-\Delta t) + 6f(t-2\Delta t) - f(t-3\Delta t)\right].$$

The systems (5) and (7) can now be transcribed in systems in the $\sim N_t N_x N_y$ unknowns $a^{(k)}_{i+\frac{1}{2},j+\frac{1}{2}}$, $u^{(k)}_{i,j+\frac{1}{2}}$, and $v^{(k)}_{i+\frac{1}{2},j}$, see [3] for details. Most of the time in Algorithm 3.1 is spent in step 5 in the solution of the linear system (7). The straightforward second-order central difference $\frac{1}{2\Delta t}(f(t+\Delta t) - f(t-\Delta t))$ could also be used to approximate $\partial f/\partial t$. It however has drawbacks. Most of all, it splits the problem in two independent subproblems if N_t is even.

5 Numerical Solution

The system matrix $\mathcal{H}$ in (7) has a block structure typical for three-dimensional finite difference discretizations (Fig. 3). The principal 3×3 block structure of $\mathcal{H}$ stems from the three components a, u, and v. The components corresponding to the spatial derivatives give rise to tridiagonal blocks. The components corresponding to the temporal derivative entail cyclic blocks with five bands. These bands can be seen quite well in Fig. 3 because the index in t-direction varies more slowly than the indices of the spatial directions.

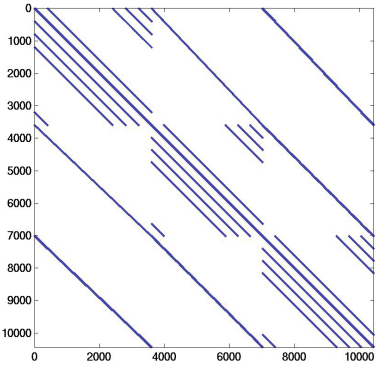


Fig. 3. MATLAB spy of $\mathcal{H}$ for $N_x = N_y = 20$ and $N_t = 9$. The most prominent 3×3 block structure stems from the three components a, u, and v. The diagonal blocks are cyclic.

As $\mathcal{H}$ is non-symmetric we solve system (7) by the GMRES algorithm [9]. The preconditioner is based on the specific structure of $\mathcal{H}$. In a first step, we approximate $\mathcal{H}$ by replacing all diagonal and off-diagonal blocks with the closest cyclic or Toeplitz blocks [1], i.e., the elements in each (off-)diagonal of a block are replaced by the average of the respective (off-)diagonal. The cyclic $N_t \times N_t$ blocks of the modified $\mathcal{H}$ can now be diagonalized by applying FFTs from left and right. There are $N_x(N_y-1) + (N_x-1)N_y + N_x N_y$ independent FFTs to be

applied from either side. Notice that the eigenvalues of the cyclic blocks come in complex conjugate pairs such that complex arithmetics is required in the following steps.

After these diagonalizations, the matrix splits in $N_t' = (N_t - 1)/2$ separate but complex spatial problems, corresponding to the individual Fourier modes $k = 0, \ldots, N_t'$. In general, the systems are still too large to be factored. Therefore, in the second step, we use preconditioned GMRES solvers to solve the N_t systems of equations in parallel. We form the preconditioner with the same recipe as before: quantities along the x-axis with equal y-coordinate are averaged to make the tridiagonal blocks Toeplitz blocks. After the diagonalization by FFTs, we arrive at $N_t \cdot N_x$ independent $N_y \times N_y$ problems which can be solved by Gaussian elimination.

6 Parallelization

Our code is parallelized making use of Trilinos [2,11], a collection of numerical software packages. We use the GMRES solver in the package AztecOO. As AztecOO can only solve real valued systems, we additionally use the package Komplex that generates an equivalent real valued problem out of a complex problem by splitting real and imaginary parts. This is not optimal, because the current implementation of this package explicitly generates the real valued system, which uses again time and memory space. One could avoid this overhead by using Belos, which is also able to solve complex systems (by means of generic programming). Additionally we use the FFTW library for the Fourier transforms and DGBTRF and DGBTRS from LAPACK to solve the banded systems.

In general, the matrices and vectors are distributed such that each process gets approximately the same number of rows. For $\mathcal{A}$ and $\mathcal{H}$, the spatial domain is divided into m_x intervals along the x-axis and m_y intervals along the y-axis. The time domain is divided into m_t intervals. Each process gets then equations corresponding to all variables in one of these $m_x m_y m_t$ blocks. Because of this partitioning we set the number of processes equal to $m_x m_y m_t$.

To construct the small systems we need to average and then to perform a Fourier transform in the time direction. To do this efficiently we need to have the entries for a certain spatial point at all time steps local in one processor's memory. So, the matrix is redistributed in "stripes", where each processor gets all rows corresponding to all time steps for some quantities. To construct the smallest systems we need to have the variables in x-direction grouped together on one processor. During the solution process, we need to have the smallest systems (each connect all y-values) local on one process, such that the (serial) factorization and forward and backward substitution can be performed. While the matrices need only to be redistributed in the beginning of each Newton iteration step, the solution vectors and right hand sides have to be redistributed prior to every outer GMRES iteration step. In the inner GMRES iterations the vectors need only to be redistributed within the group of processes that solves a certain system. Apart from the fact that each redistribution consumes time, it also uses memory.

Solving the N_t systems in parallel is not straightforward as their condition numbers vary considerably. Some of the systems are diagonally dominant while others are close to singular, cf. Fig. 7. It turned out that the iteration count of one solve is a good prediction of the iteration count of the next solve of the same system. This makes it possible to design a simple, adaptive, and effective load balancing strategy and assign fewer or more processors to a solver. Notice that the matrices split in N_y subblocks which makes the parallelization of these solvers straightforward.

7 Experiments

For the numerical experiments, we use a configuration which yields a periodic steady-state solution with a primary and a secondary wave system (Fig. 4). The primary wave system is generated by a localized oscillation at the ground in the shape of a Gaussian. These waves propagate toward the boundary of the computational domain where they are damped out by the fringe forcing such that no waves are reflected from the boundaries. The propagation of these waves is non-uniform because the waves steepen and their speed is reduced in the more shallow regions of the basin. A small submerged hill in the shallow region of the basin leads to reflections of the primary waves. This results in a secondary wave system which is centered at this hill. Figure 5 shows four snapshots of the periodic steady-state solution taken at $t = 0$, $T/4$, $T/2$, and $3T/4$, respectively.

The discretization parameters of the numerical experiments are listed in Table 1. Note that case 3 uses fewer grid points in the t-direction due to memory limitations.

Table 1. Discretization parameters for the numerical experiments

case	N_x	N_y	N_t
1	100	100	99
2	200	200	99
3	400	400	39

All numerical experiments from Table 1 converge relatively fast (Fig. 6(a)). The residual norm is reduced approximately by an order of magnitude per Newton iteration such that sufficiently converged solutions can be obtained typically within less than ten Newton steps. In the wave system with a hill we observe convergence problems at least in one case. To encounter them, we will in the future increase the robustness of the Newton iteration by backtracking (line search, damping) [8].

The numbers of outer GMRES iterations per Newton step is shown in Fig. 7a. It increases successively with each Newton step. The number of inner GMRES iterations (Fig. 7b) shows a strong dependence on the Fourier mode number k. The required work for the small mode numbers is significantly larger than for high mode numbers. This effect can be explained by the diagonal dominance of

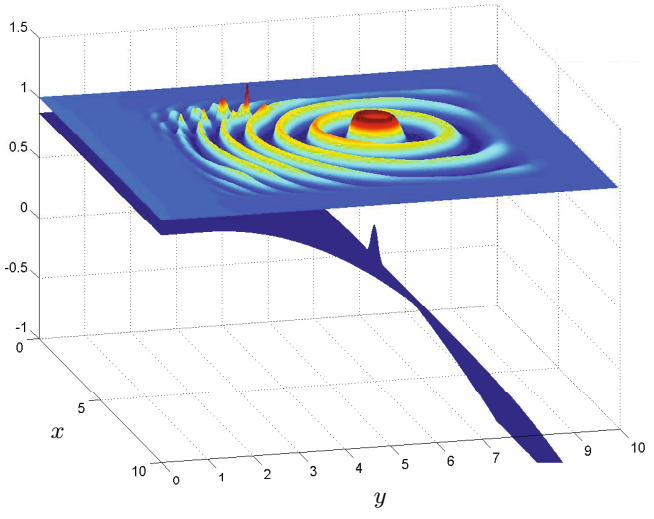
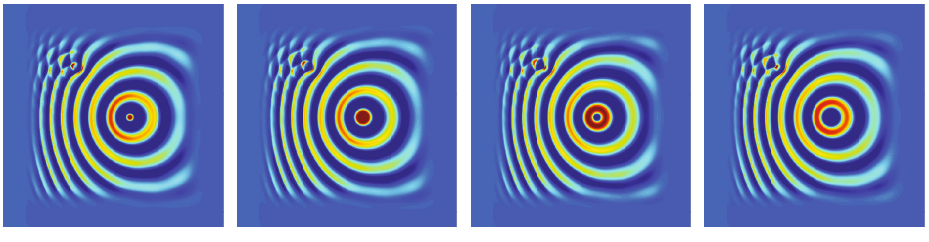


Fig. 4. Configuration for the numerical experiments with an oscillating perturbation in the center and a small submerged hill in the shallow region of the basin (top left)



(a) $t = 0$ (b) $t = T/4$ (c) $t = T/2$ (d) $t = 3T/4$

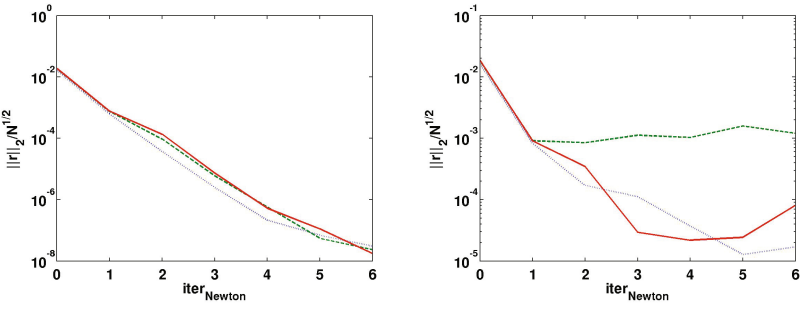
Fig. 5. Snapshots of the converged periodic solution at different phases

the inner systems of equations. The mode number k corresponds physically to a frequency and enters the inner systems as a factor on the diagonal entries. Therefore, the inner systems for high mode numbers (high frequencies, $k \approx N_t'$) are diagonally dominant and are easier to solve than the inner systems for small k.

The strong dependence between the mode number k and the number of inner GMRES iterations could lead to a strong load imbalance for the parallel tasks. Therefore, a dynamic scheduling of the inner GMRES solutions was used. It tries to balance the workload across the parallel threads. To this end, the numbers of inner iterations from the previous Newton step are used to estimate an optimal distribution of the parallel tasks to the different threads.

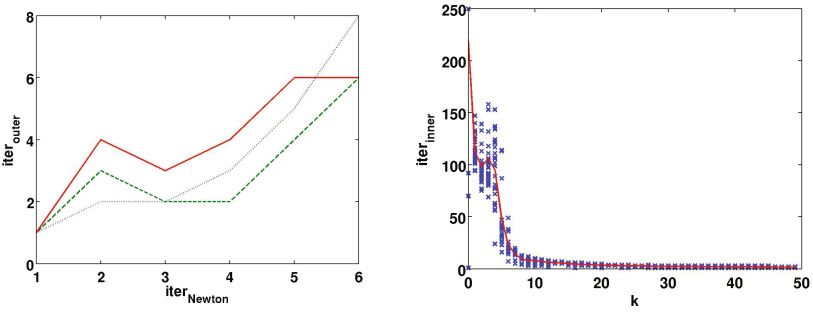
The parallel performance of the solver was measured on the large Brutus cluster at ETH Zurich[1] with AMD Opteron 8380 Quad-Core CPUs and an

[1] http://www.clusterwiki.ethz.ch/legacy/Brutus



(a) (b)

Fig. 6. Convergence of the residual norm over the Newton iterations: (a) without hill (no secondary wave system), (b) with a hill (dotted blue: case 1, dashed green: case 2, solid red: case 3)

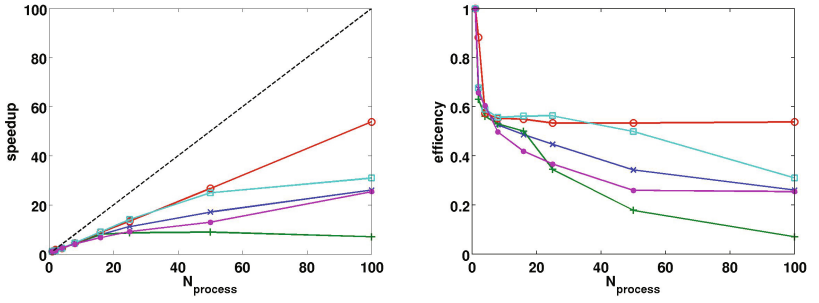


(a) (b)

Fig. 7. Number of inner/outer GMRES iterations for case 1 without hill: (a) outer GMRES iterations per Newton iteration (dotted blue: case 1, dashed green: case 2, solid red: case 3), (b) inner GMRES iterations as a function of the mode number k (blue $\times$: single instances of the inner GMRES; solid red: average over all instances of the inner GMRES)

Infiniband QDR network. Typical turn-around times on 100 cores for case 3 were approximately 430 s. In these simulations, the spatial grid was decomposed into 5 subdomains in each direction and the temporal direction was split into 4 intervals.

Figure 8 illustrates the speed-up and parallel efficiency of the solver by increasing the number of cores from 1 to 100 for the case 1. In view of the relatively small size of case 1, the relevant modules of the solver (solution, construction of the preconditioner, and update of the Newton matrix) scale reasonably well. The immediate drop in the efficiency from 1 to multiple processes is due to the reshuffling of the data to localize the FFT's which is not necessary for $N_{\text{process}} = 1$.



(a) (b)

Fig. 8. (a) Parallel speedup and (b) efficiency for the solution of case 1 (green +: initialization; red ○: update; light blue □: preconditioner construction; violet ●: solution; dark blue ×: overall; dashed black: ideal speedup)

8 Conclusions

We have presented a concept for the parallelization of a periodic shallow-water problem in space *and* in time. The presented results illustrate that the algorithm converges quickly towards the steady-state solution of the problem. Numerical experiments of different sizes show that the algorithm scales reasonably well. Nevertheless, a more efficient preconditioner would improve the performance of the present algorithm. It could be promising to further exploit the periodicity of the sought solution by replacing, for instance, the finite-difference stencil for the time derivative by a Fourier spectral method. This approach has been investigated by the authors for a one-dimensional Burgers equation [7].

The proposed algorithm should be seen in the context of the current development of supercomputers. The availability of more and more processing units will require modern solvers for fluid dynamics problems to distribute the work to ever more parallel threads. For a large class of fluid dynamics problems with (quasi-) periodic steady-state solutions, this could be achieved by parallelizing the time domain, in addition to a state-of-the-art domain decomposition in space. The classical solution procedure consists of a (often explicit Runge-Kutta) time stepping scheme that is executed for about five periods until the (quasi-) periodic solution is reached. (Here, a period refers to the time for the slowest wave to traverse the domain.) If the spatial grid is refined for accuracy reasons, the CFL stability condition requires that the time step size is reduced proportionally despite a smooth temporal behavior of the solution. Finally, parallelism is restricted to the space dimension(s).

The space-time approach requires additional memory space to store the matrix and preconditioner in the Newton step. But even if a space-time approach leads to an increased overall workload, the turn-around time of the latter can be reduced due to the increased parallelism.

References

1. Chan, T.F.: An optimal circulant preconditioner for Toeplitz systems. SIAM J. Sci. Stat. Comput. 9, 766–771 (1988)
2. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the Trilinos project. ACM Trans. Math. Softw. 31(3), 397–423 (2005)
3. Hiltebrand, A.: Parallel solution of time-periodic problems. Master thesis, ETH Zurich, Institute of Fluid Dynamics (March 2011)
4. Kevorkian, J.: Partial Differential Equations: Analytical Solution Techniques, 2nd edn. Springer, New York (2000)
5. LeVeque, R.J.: Finite Difference Methods for Ordinary and Partial Differential Equations. SIAM, Philadelphia (2007)
6. Lions, J.-L., Maday, Y., Turinici, G.: A "parareal" in time discretization of PDE's. C. R. Math. Acad. Sci. Paris 332(7), 661–668 (2001)
7. Obrist, D., Henniger, R., Arbenz, P.: Parallelization of the time integration for time-periodic flow problems. PAMM 10(1), 567–568 (2010)
8. Pawlowski, R.P., Shadid, J.N., Simonis, J.P., Walker, H.F.: Globalization techniques for Newton–Krylov methods and applications to the fully coupled solution of the Navier–Stokes equations. SIAM Rev. 48(4), 700–721 (2006)
9. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
10. Stoll, M., Wathen, A.: All-at-once solution of time-dependent PDE-constrained optimization problems. Technical Report 10/47, Oxford Centre for Collaborative Applied Mathematics, Oxford, England (2010)
11. The Trilinos Project Home Page, http://trilinos.sandia.gov/
12. Vreugdenhil, C.B.: Numerical Methods for Shallow-Water Flow. Kluwer, Dordrecht (1994)

A Parallel 3D Unstructured Implicit RANS Solver for Compressible and Incompressible CFD Simulations

Aldo Bonfiglioli[1], Sergio Campobasso[2],
Bruno Carpentieri[3], and Matthias Bollhöfer[4]

[1] Dip.to di Ingegneria e Fisica dell'Ambiente, University of Basilicata, Potenza, Italy
aldo.bonfiglioli@unibas.it
[2] School of Engineering, Glasgow University, Glasgow G12 8QQ, UK
sergio.campobasso@glasgow.ac.uk
[3] Institute of Mathematics and Computing Science, University of Groningen,
9700 AK Groningen, The Netherlands
b.carpentieri@rug.nl
[4] Institute of "Computational Mathematics",
Technische Universität Braunschweig, Germany
m.bollhoefer@tu-bs.de

Abstract. A recently developed parallel three-dimensional flow solver of the turbulent Navier-Stokes equations, based on Fluctuation Splitting discretization scheme and an effectively preconditioned Newton-Krylov subspace algorithm, is presented. The high computational performance is demonstrated using a suite of two- and three-dimensional test cases with grids featuring up to 6.5 million nodes.

Keywords: Computational Fluid Dynamics, Fluctuation splitting discretization schemes, Newton's methods, Krylov subspace solvers.

1 Introduction

This paper presents the algorithmic and parallel features of EulFS, an implicit Reynolds Averaged Navier Stokes equations (or, briefly, RANS) solver based on a Fluctuation Splitting (FS) space discretization scheme and using a preconditioned Newton-Krylov method for the integration. Although explicit multigrid techniques have dominated the CFD arena for a long time, implicit methods based on Newton's rootfinding algorithm are recently receiving increasing attention because of their potential to converge in a very small number of iterations. However, implicit CFD solvers require the implementation of well-suited convergence acceleration techniques in order to be competitive with more conventional solvers in terms of CPU cost [8]. In this study, special care is devoted to the development of critical features of the implementation, such as the choice of the preconditioning strategy for inverting the large nonsymmetric linear system at each step of the Newton's algorithm. The choice of the linear solver and the

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 313–322, 2012.

preconditioner can have a strong impact on the computational efficiency, especially when the mean flow and turbulence transport equations are solved in fully coupled form, like we do in EulFS.

2 Governing Equations

Throughout this paper, we use standard notation for the kinematic and thermodynamic variables: $\boldsymbol{u}$ is the flow velocity, ρ is the density, p is the pressure, T is the temperature, e and h are respectively the specific total energy and enthalpy, ν is the laminar kinematic viscosity and $\tilde{\nu}$ is a scalar variable related to the turbulent eddy viscosity via a damping function. The sound speed a is the square root of the artificial compressibility parameter. In the simulations this parameter is set equal to the freestream velocity. For compressible flows the sound speed a is a function of temperature while for incompressible flows it is taken constant and equal to the free-stream velocity. The mesh is partitioned into nonoverlapping control volumes, drawn around each gridpoint by joining in two space dimensions the centroids of gravity of the surrounding cells with the midpoints of all the edges that connect that gridpoint with its nearest neighbors.

In the case of high Reynolds number flows, we account for turbulence effects by the RANS equations that are obtained from the Navier-Stokes (NS) equations by means of a time averaging procedure. The RANS equations have the same structure as the NS equations with an additional term, the Reynolds' stress tensor, that accounts for the effects of the turbulent scales on the mean field. Using Boussinesq's approximation to link the Reynolds' stress tensor to the mean velocity gradient through the turbulent (or eddy) viscosity, the RANS equations become formally identical to the NS equations, except for an "effective" viscosity (and thermal conductivity), sum of the laminar and eddy viscosities (and similarly for the laminar and turbulent termal conductivity), which appears in the viscous terms. In the present study, the turbulent viscosity is modeled using the Spalart-Allmaras one-equation model [7].

Given a control volume C_i, fixed in space and bounded by the control surface ∂C_i with inward normal $\boldsymbol{n}$, we write the governing conservation laws of mass, momentum, energy and turbulence transport equations as:

$$\int_{C_i} \frac{\partial \boldsymbol{q}_i}{\partial t}\, dV = \oint_{\partial C_i} \boldsymbol{n} \cdot F\, dS - \oint_{\partial C_i} \boldsymbol{n} \cdot G\, dS + \int_{C_i} \boldsymbol{s}\, dV, \tag{1}$$

where we denote by $\boldsymbol{q}$ the vector of conserved variables. For compressible flows, we have $\boldsymbol{q} = (\rho, \rho e, \rho \boldsymbol{u}, \tilde{\nu})^T$, and for incompressible, constant density flows, $\boldsymbol{q} = (p, \boldsymbol{u}, \tilde{\nu})^T$. In (1), the operators F and G represent the inviscid and viscous fluxes, respectively. For compressible flows, we have

$$F = \begin{pmatrix} \rho \boldsymbol{u} \\ \rho \boldsymbol{u} h \\ \rho \boldsymbol{u}\boldsymbol{u} + p\mathbf{I} \\ \tilde{\nu} \boldsymbol{u} \end{pmatrix}, \quad G = \frac{1}{\mathrm{Re}_\infty} \begin{pmatrix} 0 \\ \boldsymbol{u} \cdot \tau + \nabla q \\ \tau \\ \frac{1}{\sigma} \left[(\nu + \tilde{\nu}) \nabla \tilde{\nu} \right] \end{pmatrix},$$

and for incompressible, constant density flows,

$$F = \begin{pmatrix} a^2 \boldsymbol{u} \\ \boldsymbol{u}\boldsymbol{u} + p\mathbf{I} \\ \tilde{\nu}\boldsymbol{u} \end{pmatrix}, \quad G = \frac{1}{\mathrm{Re}_\infty} \begin{pmatrix} 0 \\ \tau \\ \frac{1}{\sigma}\left[(\nu + \tilde{\nu})\,\nabla\tilde{\nu}\right] \end{pmatrix},$$

where τ is the Newtonian stress tensor. The source term vector $\boldsymbol{s}$ has a non-zero entry only in the row corresponding to the turbulence transport equation, which takes the form

$$c_{b1}\left[1 - f_{t2}\right]\tilde{S}\tilde{\nu} + \frac{1}{\sigma Re}\left[c_{b2}\left(\nabla\tilde{\nu}\right)^2\right] + \; - \frac{1}{Re}\left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left[\frac{\tilde{\nu}}{d}\right]^2. \qquad (2)$$

For a description of the various functions and constants involved in (2) we refer the reader to [7].

3 Solution Techniques

In the fluctuation splitting approach, the integral form of the governing equations (1) is discretized over each control volume C_i evaluating the flux integral over each triangle (or tetrahedron) in the mesh, and then splitting it among its vertices [4]. Therefore, we may write Eq. (1)

$$\int_{C_i} \frac{\partial \boldsymbol{q}_i}{\partial t}\, dV = \sum_{T \ni i} \boldsymbol{\phi}_i^T$$

where

$$\boldsymbol{\phi}^T = \oint_{\partial T} \boldsymbol{n} \cdot F\, dS - \oint_{\partial T} \boldsymbol{n} \cdot G\, dS + \int_T \boldsymbol{s}\, dV$$

is the flux balance evaluated over cell T and $\boldsymbol{\phi}_i^T$ is the fraction of cell residual scattered to vertex i. The space discretization of the governing equations leads to a system of ordinary differential equations:

$$M\frac{d\boldsymbol{q}}{dt} = \boldsymbol{r}(\boldsymbol{q}), \qquad (3)$$

where t denotes the pseudo time variable, M is the mass matrix and $\boldsymbol{r}(\boldsymbol{q})$ represents the nodal residual vector of spatial discretization operator, which vanishes at steady state. The residual vector is a (block) array of dimension equal to the number of meshpoints times the number of dependent variables, m; for a one-equation turbulence model, $m = d + 3$ for compressible flows and $m = d + 2$ for incompressible flows, d being the spatial dimension. If the time derivative in equation (3) is approximated using a two-point one-sided finite difference (FD) formula we obtain the following implicit scheme:

$$\left(\frac{1}{\Delta t^n}V - J\right)\left(\boldsymbol{q}^{n+1} - \boldsymbol{q}^n\right) = \boldsymbol{r}(\boldsymbol{q}^n), \qquad (4)$$

where we denote by J the Jacobian of the residual $\dfrac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}}$. Eq. (4) represents a large nonsymmetric sparse linear system of equations to be solved at each pseudo-time step for the update of the vector of the conserved variables. The nonzero pattern of the sparse coefficient matrix is symmetric, *i.e.* entry (i, j) is nonzero if and only if entry (j, i) is nonzero as well; on average, the number of non-zero (block) entries per row equals 7 in 2D and 14 in 3D. The analytical evaluation of the Jacobian matrix though not impossible, is rather cumbersome. In a practical implementation we adopt a two-step approach. In the early stages of the calculation, the RANS equations are solved in a loosely coupled fashion: the mean flow solution is advanced over a single time step using an approximate (Picard) Jacobian while keeping turbulent viscosity frozen, then the turbulent variable is advanced over one or more pseudo-time steps using a FD Jacobian with frozen mean flow variables. This procedure will eventually converge to steady state, but never yields quadratic convergence. Once the solution estimate has become sufficiently close to the sought steady state solution, which can be tested by monitoring the norm of the nodal residual $\boldsymbol{r}$, we let Δt^n grow unboundedly so that Newton's method is recovered during the last steps of the iterative process. The time step length is selected according to the Switched Evolution Relaxation (SER) strategy as:

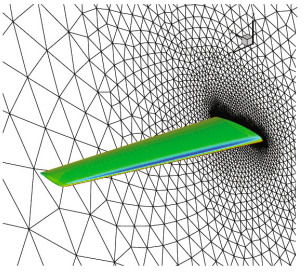
$$\Delta t^n = \Delta t \min\left(C_{\max}, C_0 \frac{||\boldsymbol{r}(\boldsymbol{q}^{n+1,0})||_2}{||\boldsymbol{r}(\boldsymbol{q}^{n+1,k})||_2}\right), \tag{5}$$

where Δt is the pseudo time-step based upon the stability criterion of the explicit time integration scheme; C_0 and $C_{\max}$ are user defined constants that control the initial and maximum pseudo time-steps used in the actual calculations.

4 3D Incompressible Turbulent Flow Simulations Past the DPW3 Wing-1

This section presents the turbulent incompressible flow analysis of a three-dimensiopnal wing. The geometry, illustrated in Fig. 1, was proposed in the 3rd Drag Prediction Workshop; we refer to this geometry as the "DPW3 Wing-1". Flow conditions are 0.5° angle of attack and Reynolds number based on the reference chord equal to $5\,10^6$. The freestream turbulent viscosity is set to ten percent of its laminar value.

Typical convergence histories obtained running the loosely coupled approach in EulFS are displayed in Figs. 2(a) and 2(c). Ten iterations were performed on the turbulence transport equation for each mean flow iteration. The residual histories are characterized by an initial transient stage during which the residual of the conservation equations remains almost constant, whereas the residual of the turbulence transport equation shows ample oscillations. During this first stage, turbulent viscosity builds up starting from its low, initial value. Afterwords, all residuals start converging steadily towards machine zero. During the second stage, the residuals undergo a reduction of about seven orders of magnitude in



Ref. Area, S = 290322 mm^2 = 450 in^2
Ref. Chord, c = 197.556 mm = 7.778 in
Ref. Span, b = 1524 mm = 60 in
Mesh1: Larc Medium 4,476,969 grid pts
Mesh2: Cessna Fine 6,138,245 grid pts

Fig. 1. Geometry and mesh characteristics of the DPW3 Wing-1 problem

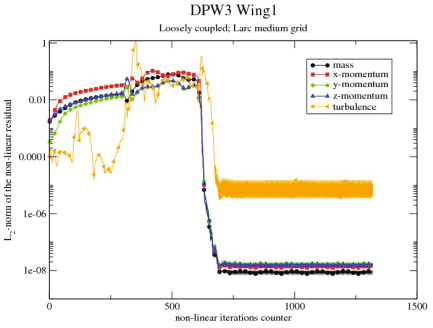
about 100 outer (*i.e.* nonlinear) iterations. Notice, however, that the loosely coupled solution strategy may not be always capable to drive the nodal residuals to machine zero. This is the third stage of the convergence, which is typically observed when using the loosely coupled solution strategy; it is clearly visible in Figs. 2(a) and 2(c). Using the solution obtained at the end of stage three as the initial guess for the fully coupled approach using Newton's algorithm, a very limited number (of the order of ten) of outer iterations is generally needed to bring all residuals down to machine accuracy, as shown in Figs. 2(b) and 2(d).

In all our experiments, the inner iterative solvers was GMRES(30) [6], i.e. re-started every 30 inner iterations. On the finest available grid, see Fig. 2(d), we tested various preconditioners available in the Petsc library [1] for GMRES, including block Jacobi (BJ) and Additive Schwarz (ASM) with one or two levels of overlap. An Incomplete Lower Upper ILU(l) factorization of level l was applied to each subdomain. The results are summarized in Table 1. The column o-it gives the number of outer iterations required by each preconditioner combination to drive the mass conservation residual below 10^{-12} whereas the column i-it gives the average number of iterations needed by the iterative solver to meet the convergence criterion. In PETSc iterations are terminated when

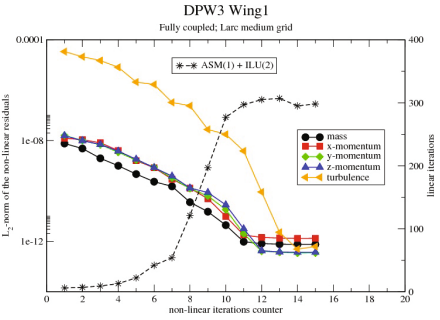
$$||\boldsymbol{b} - A\boldsymbol{x}_k||_2 < \max\left(10^{-5}||\boldsymbol{b}||_2, 10^{-50}\right) \tag{6}$$

and $\boldsymbol{x}_0$ is the zero vector. The column labeled CPU reports the CPU solution time in seconds required by each run. Finally, the "fill ratio" measures the number of non-zero entries in the ILU(l) factors with respect to those of the Jacobian matrix and represents the memory occupation. All calculations were run on the Matrix cluster located at the inter-university computing center CASPUR (Rome, Italy). The Matrix cluster consists of 258 nodes, each equipped with two AMD Opteron quadcore processors and interconnected with an InfiniBand technology (20Gbit/sec). We used 72 cores to run the simulation on the Larc medium, and 96 cores on the Cessna Fine.

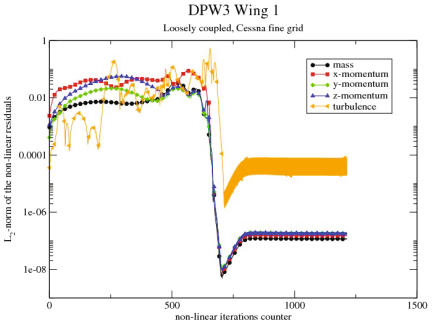
The number of inner iterations greatly varied depending on the length the pseudo time step Δt^n in Eq. (4), as visible in Figs. 2(b) and 2(d). There are two possible causes of this phenomenon. Close to the steady-state, the convergence criterion defined by Eqn. (6) becomes tighter because $\boldsymbol{b} = \boldsymbol{r}(\boldsymbol{q})$ vanishes. Also,



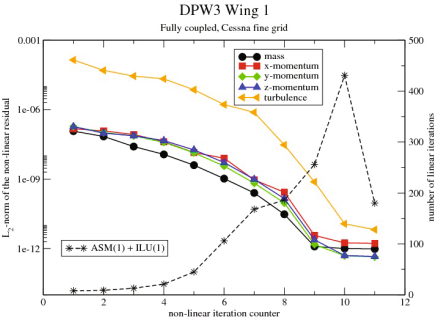
(a) Larc medium mesh: loosely coupled solution strategy.



(b) Larc medium mesh: fully coupled solution strategy.



(c) Cessna fine mesh: loosely coupled solution strategy.



(d) Cessna fine mesh: fully coupled solution strategy.

Fig. 2. Convergence histories for the DPW3 Wing-1 problem

using very large steps the pseudo time term $1/\Delta t^n$ in Eq. (4) vanishes and the diagonal dominance of the linear system (4) being solved decreases making the construction of a numerically stable ILU factorization more problematic.

Concerning the performance of the considered preconditioners, the basic ILU(0) preconditioner did not enable the code to converge within the 1500 preset maximum number of inner GMRES iterations using either BJ or ASM. This happened because the number of inner GMRES iterations required to achieve convergence significantly increased when the pseudo-time term in Eq. (4) started to decrease approaching zero. All present convergence requirements could instead be met by using at least one level of fill. The best performance in terms of CPU-time was achieved by using one level of fill: the BJ+ILU(1) combination was the fastest, and slightly cheaper in terms of memory occupation than ASM(1)+ILU(1) even if the latter required less inner iterations; doubling the level of overlap in ASM reduced the number of inner iterations, but slightly increased the CPU cost. Augmenting further the pattern of ILU(l) helped to reduce the average number of inner iterations, but also increased the solution and memory costs of our solver. The last column of Tab. 1 gives the percentage ratio between the time spent solving the linear system and the total running time averaged over all outer iterations. For the

best preconditioner combinations in the case under study the time spent solving the linear system was about the same as that required to compute the residual vector and to assemble the Jacobian matrix. However, this cost may vary depending on the physical model being used, whether compressible or incompressible, and the features of the mesh, such as the presence of high aspect ratio tetrahedra.

Table 1. Performance of selected preconditioners on the DPW3 Wing-1 (fine Cessna mesh) problem

pc	l	fill ratio	o-it	CPU	i-it	ksp/total %
BJ	0	1	n.c.	3247.0	654	79.6
BJ	1	2.00818	10	1515.0	133	51.4
BJ	2	4.03536	11	1933.0	101	56.8
BJ	3	6.89511	11	2409.0	83	65.6
ASM(1)	1	0	n.c.	3539.0	699	81.6
ASM(1)	1	2.0994	11	1806.0	134	55.9
ASM(1)	2	4.40286	11	2018.0	91	59.9
ASM(2)	1	2.06889	11	1879.0	128	58.2
ASM(2)	2	4.25357	11	2116.0	86	62.3

5 Experiments with Multilevel Preconditioning

Notwithstanding their ease of parallelization, BJ and ASM methods have limited algorithmic scalability. The data of Table 2 examine the parallel performance of EulFS on the Matrix cluster located at the CASPUR computing center. We report the total solution time spent to solve the DPW3 Wing-1 problem, the elapsed time for solving the whole sequence of linear systems required by the Newton's iterations, and the average number of GMRES iterations. For consistency of results, the statistics were obtained averaging on three runs. In our experiment on 224 cores, we observed that more than 70% of the computational time was spent on solving linear systems, whereas only 55% on 96 cores. The quasi-linear increase of the number of inner GMRES iterations, reported in column four of Table 2, was mainly responsible of the overall lack of parallel scalability.

Prompted by the need to enhance both the robustness and the scalability of the linear solver in EulFS, we are investigating multilevel preconditioners based on ILU factorization beside the preconditioners already available in PETSc. To simplify the notation in the remainder of this Section, we first re-write the sparse linear system to be solved at each Newton step as:

$$Ax = b. \tag{7}$$

Following [2], we initially rescale and reorder the initial matrix A as

$$P^T D_l A D_r Q = \hat{A} \tag{8}$$

Table 2. Scalability results of the EulFS code on the DPW3 Wing-1 (fine Cessna mesh) problem. The number of linear iterations is averaged over three runs, as we observed little differences due to round-off errors.

Number of cores	Total simulation time (sec)	Cumulative time linear solver (sec)	Average number linear iterations
96	1542	839	133.1
128	1418	864	139.4
160	1388	927	210.9
224	1176	853	250.7

which yields $\hat{A}\hat{x} = \hat{b}$ for appropriate $\hat{x}, \hat{b}$. The initial step may consist of an optional maxium weight matching [5].

By rescaling and performing a one-sided permutation, one attempts to improve the diagonal dominance. Thereafter a symmetric reordering is applied to reduce the fill-in bandwidth. The symmetric reordering can also be used without an a priori matching step, only rescaling the entries and symmetrically permuting the rows and columns. This is of particular interest for (almost) symmetrically structured problems. Next, an inverse-based ILU factorization with static diagonal pivoting is computed. More precisely, during the approximate incomplete factorization $\hat{A} \approx LDU$ with L and U^T being unit lower triangular factors and D a block diagonal matrix, the norms $\|L^{-1}\|$, $\|U^{-1}\|$ are estimated. If at factorization step l a prescribed bound κ is exceeded, the current row l and column l are permuted to the lower right end of the matrix. Otherwise the approximate factorization is continued. One single pass leads to an approximate partial factorization

$$\Pi^T \hat{A} \Pi = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \approx \begin{pmatrix} L_B & 0 \\ L_E & I \end{pmatrix} \begin{pmatrix} D_B & 0 \\ 0 & S_C \end{pmatrix} \begin{pmatrix} U_B & U_F \\ 0 & I \end{pmatrix} \equiv L_1 D_1 U_1, \qquad (9)$$

with a suitable leading block B and a suitable permutation matrix Π, where $\|L_1^{-1}\| \leq \kappa$, $\|U_1^{-1}\| \leq \kappa$. The remaining system S_C approximates $C - EB^{-1}F$ and from the relations

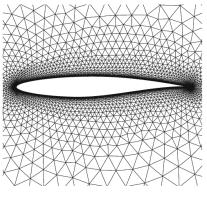
$$\begin{cases} B\hat{x}_1 + F\hat{x}_2 = \hat{b}_1 \\ E\hat{x}_1 + C\hat{x}_2 = \hat{b}_2 \end{cases} \Rightarrow \begin{cases} \hat{x}_1 = B^{-1}(\hat{b}_1 - F\hat{x}_2) \\ (C - EB^{-1}F)\hat{x}_2 = \hat{b}_2 - EB^{-1}\hat{b}_1 \end{cases},$$

at each step of an iterative solver we need to store and invert only blocks with B and $S_C \approx C - EB^{-1}F$ while for reasons of memory efficieny, L_E, U_F are discarded and implicitly represented via $L_E \approx EU_B^{-1}D_B^{-1}$ (resp. $U_F \approx D_B^{-1}L_B^{-1}F$). When the scaling, preordering and the factorization is successively applied to S_C, a multilevel variant of (8) is computed. The multilevel algorithm ends at some step m when either S_C is factored completely or it becomes considerably dense and switches to a dense LAPACK solver. After computing an m-step ILU decomposition, for preconditioning we have to apply $L_m^{-1}AU_m^{-1}$. From the error equation $E_m = A - L_m D_m U_m$, we see that $\|L_m^{-1}\|$ and $\|U_m^{-1}\|$ contribute to the inverse

error $L_m^{-1}E_mU_m^{-1}$. Monitoring the growth of these two quantities during the partial factorization is essential to preserve the numerical stability of the solver.

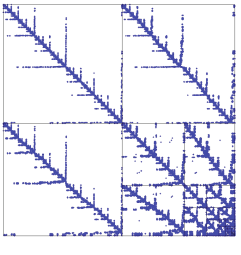
The test-case that we consider is the compressible, subsonic flow past the two-dimensional RAE2822 profile. Free-stream conditions are as follows: Mach number $M_\infty = 0.676$, Reynolds' number based on chord: $\mathrm{Re}_C = 5.7 \cdot 10^6$, angle of attack: $\alpha_\infty = 2.40°$. The computational mesh, which is shown in Figure 3, is made of 10599 meshpoints and 20926 triangles. The simulation was started from uniform flow and the solution advanced in pseudo-time using the approximate linearization. Once the L_2 norm of the residual was reduced below a preset threshold, the fully coupled approach was put in place.

In the table shown in Figure 3, we report the number of iterations of GMRES(30) to reduce the initial residual by five orders of magnitude, starting from the zero vector. The multilevel ILU needed five levels of factorization and we chose a drop tolerance of 10^{-2} for both the L, U factors and the final Schur complement matrix. In our runs, we tested different choices of static ordering (cf. Figure 4) available in the ILUPACK software package [3]: the Reverse Cuthill-McKee, Minimum Degree orderings and the Multilevel Nested Dissection were the most efficient, and compared equally well without Maximum Weight Matching, whereas

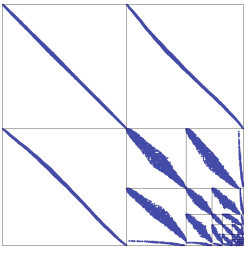


Ordering	$\frac{nnz(M)}{nnz(A)}$	#GMRES(30)
amd	5.07	23
metisn	5.47	19
metise	5.54	18
rcm	5.77	18
mmd	5.36	18
amf	4.98	19

Fig. 3. Geometry of the RAE2822 profile and performance of the multilevel linear solver. Acronyms for the built-in reordering schemes available in the ILUPACK package: *amd*=Approximate Minimum Degree, *metisn*=Metis Multilevel Nested Dissection by nodes, *metise*=Metis Multilevel Nested Dissection by edges, *rcm*=Reverse Cuthill-McKee, *mmd*=Minimum Degree, *amf*=Approximate Minimum fill.



(a) Nested Dissection ordering.



(b) Reverse Cuthill-McKee ordering.

Fig. 4. Nonzero pattern of the multilevel preconditioner on the RAE2822 airfoil

including this option, it did not enable us to converge within 500 iterations. We observed that for reasonable value of the density of the preconditoner we could achieve fast and stable convergence in GMRES, at low restart. The overall results show that the proposed multilevel framework may be well-suited to use for solving realistic (possibly large) compressible turbulent flow problems in Newton-Krylov formulations.

6 Conclusions

In this paper we have illustrated the algorithmic and parallel features of a parallel three-dimensional flow solver of the turbulent Navier-Stokes equations. The presented CFD solver has a very high computational performance, that is highlighted by experiments on the calculation of the flow past the DPW3 Wing-1 configuration of the $3^r d$ AIAA Drag Prediction Workshop. Preliminary encouraging experiments are shown with multilevel preconditioning for enhancing the robustness of the inner linear solver. The theoretical and numerical results reported in this study will contribute to highlight the potential and enrich the database of implicit solvers in CFD simulations.

References

1. Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., Curfman McInnes, L., Smith, B.F., Zhang, H.: PETSc home page (2001), http://www.mcs.anl.gov/petsc
2. Bollhöfer, M., Saad, Y.: Multilevel preconditioners constructed from inverse–based ILUs. SIAM J. Scientific Computing 27(5), 1627–1650 (2006)
3. Bollhöfer, M., Saad, Y., Schenk, O.: ILUPACK — preconditioning software package. Release 2.3 (December 2010), http://ilupack.tu-bs.de/
4. Bonfiglioli, A.: Fluctuation splitting schemes for the compressible and incompressible Euler and Navier-Stokes equations. IJCFD 14, 21–39 (2000)
5. Duff, I.S., Koster, J.: The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. SIAM J. Matrix Analysis and Applications 20(4), 889–901 (1999)
6. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing, New York (1996)
7. Spalart, P.R., Allmaras, S.R.: A one-equation turbulence model for aerodynamic flows. La Recherche-Aerospatiale 1, 5–21 (1994)
8. Wong, P., Zingg, D.W.: Three-dimensional aerodynamic computations on unstructured grids using a Newton-Krylov approach. Computers and Fluids 37, 107–120 (2008)

Parallelization of the Discrete Chaotic Block Encryption Algorithm

Dariusz Burak and Michał Chudzik

Faculty of Computer Science and Information Technology
West Pomeranian University of Technology, ul.Żołnierska 49, 71-210 Szczecin, Poland
{dburak,mchudzik}@wi.zut.edu.pl

Abstract. In this paper, we present the results of parallelizing the Lian et al. discrete chaotic block encryption algorithm. The data dependence analysis of loops was applied in order to parallelize this algorithm. The OpenMP standard is chosen for presenting the parallelism of the algorithm. We show that the algorithm introduced by Lian et al. can be divided into parallelizable and unparallelizable parts. As a result of our study, it was stated that the most timeconsuming loops of the algorithm are suitable for parallelization. The efficiency measurement for a parallel program is presented.

Keywords: parallelization, encryption, chaos-based algorithm, data dependency analysis, OpenMP.

1 Introduction

One of the most important functional features of cryptographic algorithms is a cipher speed. This feature is extremely important in case of block ciphers because they usually work with large data sets. Thus even not much differences of speed may cause the choice of the faster cipher by the user. Therefore, it is so important to parallelize encryption algorithms in order to achieve faster processing using multi-core processors and multi-processor systems. In recent years, besides classical block ciphers such as Rijndael, Camellia or International Data Encryption Algorithm (IDEA), alternative approaches of constructing ciphers based on the application of the theory of chaotic dynamical systems has been developed. Nowadays, there are many descriptions of various block ciphers based on chaotic maps, for instance [1], [2], [3], [4], [5], [6]. The important issue in chaotic ciphers is program implementation. Unlike parallel implementations of classical block ciphers, for instance Rijndael [7], IDEA [8], there is no parallel implementations of chaotic block ciphers. It looks like a research gap because only software or hardware implementation will show real functional advantages and disadvantages of encryption algorithms. Considering this fact, the main contribution of the study is developing a parallel algorithm in accordance with OpenMP of the well-known Lian et al. cipher (called further LSW cipher) [9] based on the transformations of a source code written in the C language representing the sequential algorithm.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 323–332, 2012.

2 The LSW Encryption Algorithm

The LSW algorithm was published by Lian et al. in 2005 [9]. It is an interesting example of a discrete chaotic block cipher. It is based on chaotic standard maps used as components for a confusion process, diffusion function, key generation and distribution. The structure of the LSW cipher is shown in Fig. 1.

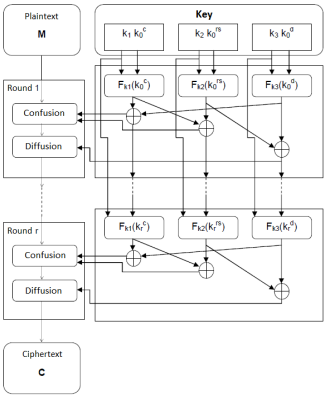


Fig. 1. Structure of the LSW encryption algorithm

The encryption and the decryption process is shown in Fig. 2, where P is a plaintext, C is a ciphertext, K_c is a key for confusion process, K_d is a key for a diffusion function.

The confusion process $C_e(P_i, K_{ci})$ is firstly repeated for n times. Then together with the diffusion function $D_e(M_i, K_{di})$ are repeated for m times.

P_i, M_i and C_i are the plaintext, intermediate text and ciphertext of the i^{th} encryption process, respectively; m is the number of repetitions.

The encryption process is performed as follows:

$$\begin{cases} P_{i+1} = C_i(P_0 = P; i = 0, 1, ..., m-1) \\ C_i = D_e(M_i, K_{di}) = D_e(C_e^n(P_i, K_{ci}), K_{di}) \end{cases}$$

This process is repeated for m times.

Similarly, the decryption process is performed in the following way:

$$\begin{cases} C_{i+1} = P_i(C_0 = C; i = 0, 1, ..., m-1) \\ P_i = C_d^n(M_i, K_{ci}) = C_d^n(D_d, (C_i, K_{di}), K_{ci}) \end{cases},$$

where K_{ci} and K_{di} are the i^{th} confusion and diffusion key, respectively.

Encryption and decryption processes are symmetric; the encryption and decryption keys are the same.

The diffusion function is defined as follows:

$$\begin{cases} c_{-i} = K_{di} \\ c_k = p_k \oplus q[f(c_{k-1}, L] \end{cases},$$

where p_k is the k^{th} pixel in M_i, c_k is the k^{th} diffused pixel, L is the amplitude of each pixel, c_{-1} is the original value of the diffusion function.

The inverse diffusion function is the following:

$$\begin{cases} c_{-i} = K_{di} \\ p_k = c_k \oplus q[f(c_{k-1}, L] \end{cases},$$

where parameters are the same as the ones defined for the diffusion function.

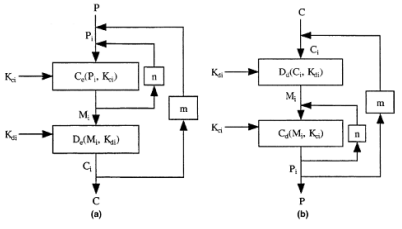


Fig. 2. Encryption and decryption processes of the LSW algorithm

More detailed description of the LSW cipher is given in [9] or [10].

3 Implementation Details of the Parallelization Process

It is necessary to prepare a C source code representing the sequential algorithm working in the Elecronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Counter (CTR) modes of operation before we start parallelizing of the LSW encryption algorithm. The source code of the algorithm in the essential ECB mode contains twenty two "for" loops including no I/O function.

In order to find dependences in program loops we have chosen Petit developed at the University of Maryland under the Omega Project and freely available for both DOS and UNIX systems. Petit is a research tool for analyzing array data dependences [11], [12].

In order to present parallelized loops, we have used the OpenMP standard. The OpenMP Application Program Interface (API) [13], [14] supports multi-platform shared memory parallel programming in C/C++ and Fortran on all architectures including Unix and Windows NT platforms.

The process of the LSW encryption algorithm parallelization can be divided into the following stages:

1. carrying out the dependence analysis of a sequential source code in order to detect parallelizable loops,
2. selecting parallelization and code transformation methods,
3. constructing parallel forms of program loops in accordance with the OpenMP standard.

There are the following basic types of the data-dependences that occur in "for" loops [15], [16], [17]: a Data Flow Dependence , a Data Anti-dependence, an Output Dependence.

At the beginning of the parallelization process of the LSW algorithm, we carried out experiments with the sequential LSW algorithm for a 5 megabytes input file in order to find the most time-consuming functions of this algorithm. It appeared that the algorithm has two computational bottlenecks enclosed in the functions lsw_enc() and lsw_dec(). We created the lsw_enc() function in order to to enable enciphering the whichever number of data blocks and the lsw_dec() one for deciphering (by analogy with similar functions included in the C source code of the classic cryptographic algorithms like DES- the des_enc(), the des_dec(), LOKI91- the loki_enc(), the loki_dec or IDEA- the idea_enc(), the idea_dec() presented in [18]).

The most time-consuming program loops included in these functions (the first one is included in the lsw_enc() function, the second one is enclosed in the lsw_dec()) are of the following forms:

3.1 Loop of the encryption process

```
for(l=0;l< PARALLELITY;l++){
    copy(wskk,block[l]);
    for (i=1;i<=iter;i++){
        generatekey(&newkeyy,&startkey,i,iter);
        randx=(int)newkeyy.randominitial*B_S;
        randy=(int)(newkeyy.randominitial*2*B_S)%B_S;
        permutation(block2[l],wskk,randx,randy);
        diffusion(block2[l], newkeyy.diffinitial);
        copy(wskk,block2[l]);
    }
}.
```

3.2 Loop of the decryption process

```
for(l=0;l< PARALLELITY;l++){
    copy(wskk,block[l]);
    for (i=iter;i>=1;i--){
        generatekey(&newkeyy,&startkey,i,iter);
        randx=(int)newkeyy.randominitial*B_S;
        randy=(int)(newkeyy.randominitial*2*B_S)%B_S;
        invdiffusion(wskk, newkeyy.diffinitial);
        invpermutation(block2[l],wskk,randx,randy);
        copy(wskk,block2[l]);
    }
}.
```

The declaration of constants is as follows:

```
#define B_S 100 //denotes the block size( N x N)
#define PARALLELITY //denotes number of processors.
```

The declaration of variables is as follows:

```
int i, l, wsk=0;
unsigned char wskk[B_S][B_S], block[PARALLELITY][B_S][B_S];
unsigned char block2[PARALLELITY][B_S][B_S];
struct lswkey {
double confparam;
double confinitial;
double randomparam;
double randominitial;
double diffparam;
double diffinitial;
} startkey, newkeyy;.
```

Taking into account the form of loops *3.1* and *3.2* (the first loop calls the permutation function (the permutation()) and the diffusion function (the diffusion()), the second one does the inverse diffusion function (invdiffusion()) and the inverse permutation function (the invpermutation()); the source code of both the diffusion() and the invdiffusion(), and the permutation() and the invpermutation() is characterized by a high degree of similarity), we examine only the *3.1* loop. However, this analysis is valid also in the case of the *3.2* loop.

The actual parallelization process of the *3.1* loop consists of the four following stages:

1. filling in the *3.1* loop by the body of the functions: the copy(), the generationkey(), the permutation() and the diffusion() (otherwise, we cannot apply a data dependence analysis);
2. filling in the transformed loop by the body of the functions: the tentmap() (included in the body of the generationkey()) and the diffusion() (included in the body of the diffusion()) (in order to apply a data dependence analysis);
3. suitable variables privatization (l, i, j, k, m, n, prev, randy, randx, newkeyy, wskk, confinitial, randominitial, diffinitial, initial, tmp) using OpenMP (based on the results of data dependence analysis);
4. adding appropriate OpenMP directive and clauses (#pragma omp parallel for private() shared()).

The steps above result in the following parallel form of *3.1* loop in accordance with the OpenMP standard:

```
#pragma omp parallel for private(l, i, j, k, m, n, prev,
randy, randx, newkeyy, wskk, confinitial, randominitial,
diffinitial, initial, tmp)
shared(block2,block,startkey,iter,sinetable)
```

```
for(l=0;l< PARALLELITY;l++){
    copy(wskk,block[l]);
    for(i=1;i<=iter;i++){
        generatekey(&newkeyy,&startkey,i,iter);
        randx=(int)newkeyy.randominitial*B_S;
        randy=(int)(newkeyy.randominitial*2*B_S)%B_S;
        permutation(block2[l],wskk,randx,randy);
        diffusion(block2[l], newkeyy.diffinitial);
        copy(wskk,block2[l]);
    }
}.
```

The *3.2* loop was parallelized in the same way as the *3.1* loop.

We also parallelized the two other interesting loops. The first of them is enclosed in the permutation function (the permutation()). The parallel form of this loop according with the OpenMP standard is the following:

```
#pragma omp parallel for private(i,j,k,l)
shared(src,dst,randx,randy)
for (i=0;i<B_S;i++)
    for (j=0;j<B_S;j++){
        k=(int)floor((i + randx + j + randy))%B_S;
        l=(int)floor((j + randy + sinetable[k]))%B_S;
        if (l<0) l+=B_S;
        if (k<0) k+=B_S;
        dst[k][l]=src[i][j];
    }.
```

The second one is included in the inverse permutation function (the invpermutation()). The parallel form of this loop is presented bellow:

```
#pragma omp parallel for private(i,j,k,l)
shared(src,dst,randx,randy)
for (i=0;i<B_S;i++)
    for (j=0;j<B_S;j++){
        l=(int)ceil((j - (randy + sinetable[i])))%B_S;
        k=(int)ceil((i - (randx +l + randy)))%B_S;
        if (l<0) l=B_S+l;
        if (k<0) k=B_S+k;
        dst[k][l]=src[i][j];
    }.
```

4 Experimental Results

In order to study the efficiency of the presented LSW parallel code we used the computer with eight Quad-Core Intel Xeon processors E7310 - 1,60 GHz and

the Intel C++ Compiler ver. 11.0 (that supports the OpenMP 3.0). The results received for a 10 megabytes input file using two, four, eight, sixteen and thirty two processors versus the only one have shown in Table 1.

Table 1. Speed-up of the parallel LSW algorithm in the ECB mode of operation

Number of processors	Number of threads	Speed-up of the encryption process	Speed-up of the decryption process	Speed-up of the whole LSW algorithm
1	1	1.00	1.00	1.00
2	2	1.95	1.99	1.47
4	4	3.70	3.80	1.98
8	8	6.70	6.90	3.05
16	16	7.30	7.60	3.19
32	32	7.00	7.50	3.09

The total running time of the LSW algorithm consists of the following operations: data receiving from an input file, round keys generation, data encryption, data decryption, data writing to an output file (both encrypted and decrypted text).

The total speed-up of the LSW parallel algorithm depends heavily on the six factors:

1. the degree of parallelization of the loop included in the lsw_enc() function (*3.1* loop),
2. the degree of parallelization of the loop included in the lsw_dec() function (*3.2* loop),
3. the method of reading data from an input file,
4. the method of writing data to an output file,
5. the block size of the LSW encryption algorithm,
6. the number of iterations (rounds).

The results confirm that the loops included both the lsw_enc() and the lsw_dec() functions are parallelizable with high speed-up (see Table 1).

The block method of reading data from an input file and writing data to an output file was used. The following C language functions and block sizes was applied: fread() function and 1024-bytes block for data reading and fwrite() function and 512-bytes block for data writing. Using the fwrite() function is especially important; choosing, for example, the fprintf() function we got much longer time of executing our tasks.

During experiments we chose the block size equal to 100 (10 x 10) as well as four iterations. Our tests showed that these parameters provide the best encryption/decryption speed of the LSW encryption algorithm.

The parallelization of the *3.3* and *3.4* loops has only a minimal influence on the speed-up value in case of the software implementation but can be useful for hardware implementation of the parallel LSW algorithm.

Table 2. Speed-ups of the parallel LSW algorithms in the CTR, CBC and CFB mode of operation

Number of processors	Number of threads	Operation	Speed-up of the CTR mode of operation	Speed-up of the CBC mode of operation	Speed-up of the CFB mode of operation
1	1	Encryption	1.00	1.00	1.00
1	1	Decryption	1.00	1.00	1.00
2	2	Encryption	1.90	1.00	1.00
2	2	Decryption	1.95	1.95	1.95
4	4	Encryption	3.55	1.00	1.00
4	4	Decryption	3.65	3.65	3.65
8	8	Encryption	6.50	1.00	1.00
8	8	Decryption	6.70	6.60	6.60
16	16	Encryption	7.00	1.00	1.00
16	16	Decryption	7.20	7.00	7.00
32	32	Encryption	6.70	1.00	1.00
32	32	Decryption	7.00	6.70	6.80

Table 3. Speed-ups of the parallel LSW algorithm for the various plaintext (ciphertext) sizes

Number of processors	Number of threads	Operation	Plaintext / Ciphertext size [kB]								
			64	128	256	512	1024	2048	4096	8192	16384
1	1	Encryption	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	1	Decryption	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	2	Encryption	1.90	1.92	1.93	1.95	1.95	1.95	1.95	1.95	1.96
2	2	Decryption	1.96	1.96	1.96	1.97	1.97	1.98	1.98	1.99	1.99
4	4	Encryption	3.40	3.40	3.50	3.50	3.50	3.60	3.70	3.70	3.70
4	4	Decryption	3.70	3.70	3.70	3.70	3.70	3.80	3.80	3.80	3.80
8	8	Encryption	6.40	6.50	6.50	6.50	6.50	6.60	6.60	6.70	6.70
8	8	Decryption	6.70	6.80	6.80	6.80	6.80	6.90	6.90	6.90	6.90
16	16	Encryption	7.00	7.10	7.10	7.20	7.20	7.20	7.30	7.30	7.30
16	16	Decryption	7.30	7.40	7.40	7.40	7.50	7.50	7.60	7.60	7.60
32	32	Encryption	6.70	6.80	6.80	6.80	6.90	6.90	7.00	7.00	7.00
32	32	Decryption	7.30	7.30	7.30	7.40	7.40	7.40	7.50	7.50	7.50

In accordance with Amdahl's Law [19] the maximum speed-up of the LSW encryption algorithm is limited to 7.752, because the fraction of the code (the whole code) that cannot be parallelized is 0.129.

We also parallelized the LSW encryption algorithm in the CTR, CBC and CFB modes of operation (based on recommendation detailed described in [20]) (when it was possible). The results are presented in Table 2.

When the LSW algorithm operates in the ECB and CTR modes of operation, both the encryption and decryption processes are parallelizable and speed ups

of the whole algorithm are similar (see- Table 2). For the CBC and CFB modes only the decryption process is parallelized so the values of speed-up are lower than for the ECB and CTR modes of operation (see- Table 2).

In Table 3 we present speed-up of encryption and decryption processes for the various sizes of plaintext (in the case of encryption process) and ciphertext (in the case of decryption process). We modified data sizes of an input file starting from 64 kilobytes up to about 16 megabytes. In conjuction with increase of data sizes we obtained a little bit greater values of speed-ups both for encryption and decryption processes as well as for two, four, eight, sixteen and thirty two threads.

5 Conclusions

In this paper, we describe the parallelization process of the LSW algorithm which was divided into parallelizable and unparallelizable parts. We have shown that the "for" loops included in the functions responsible for the encryption and decryption processes are parallelizable.

Results obtained for encryption and decryption processes for various plaintext and ciphertext sizes (in the range from 64 KB to 16 MB) confirm that the parallelization strategy is effective and parallel enciphering and deciphering processes are scalable.

The experiments have shown that the application of the parallel LSW algorithm for multiprocessor and multi-core computers would considerably boost the time of the data encryption and decryption. We believe that the speed-ups received for these operations are satisfactory.

The parallel LSW algorithm can be also helpful for hardware implementations.

References

1. Kocarev, L., Jakimoski, G.: Logistic map as a block encryption algorithm. Physics Letters A 289(4-5), 199–206 (2001)
2. Habutsu, T., Nishio, Y., Sasase, I., Mori, S.: A Secret Key Cryptosystem Using a Chaotic Map. Trans. IEICE Japan E73(7), 1041–1044 (1990)
3. Pareek, N.K., Patidar, V., Sud, K.K.: Block cipher using 1D and 2D chaotic maps. International Journal of Information and Communication Technology 2(3) (2010)
4. Yi, X., Tan, C.H., Siew, C.K.: A new block cipher based on chaotic tent maps. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 49(12) (2002)
5. Xua, S., Wang, J., Yang, S.: A Novel Block Cipher Based on Chaotic Maps. Congress on Image and Signal Processing 3 (2008)
6. Kotulski, Z., Szczepański, J.: Discrete chaotic cryptography (DCC). In: Proc. NEEDS 1997 (1997)
7. Bielecki, W., Burak, D.: Exploiting Loop-Level Parallelism in the AES Algorithm. WSEAS Transactions on Computers 5(1), 125–133 (2006)
8. Beletskyy, V., Burak, D.: Parallelization of the IDEA Algorithm. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3036, pp. 635–638. Springer, Heidelberg (2004)

9. Lian, S., Sun, J., Wang, Z.: A Block Cipher Based on a Suitable Use of the Chaotic Standard Map. Chaos, Solitons and Fractals 26(1), 117–129 (2005)
10. Pejaś, J., Skrobek, A.: Chaos-Based Information Security. In: Handbook of Information and Communication Security, pp. 91–128 (2010)
11. Kelly, W., Maslov, V., Pugh, W., Rosser, E., Shpeisman, T., Wonnacott, D.: New User Interface for Petit and Other Extensions. User Guide (1996)
12. The Omega Project: Frameworks and Algorithms for the Analysis and Transformation of Scientific Programs, http://www.cs.umd.edu/projects/omega/
13. Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R.: Parallel Programming in OpenMP. Morgan Kaufmann Publishers, Inc. (2001)
14. OpenMP C and C++ Application Program Interface. Version 3.0 (2008)
15. Moldovan, D.I.: Parallel Processing. From Applications to Systems. Morgan Kaufmann Publishers, Inc. (1993)
16. Muchnick, S.S.: Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, Inc. (1997)
17. Allen, R., Kennedy, K.: Optimizing compilers for modern architectures: A Dependencebased Approach. Morgan Kaufmann Publishers, Inc. (2001)
18. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. John Wiley & Sons (1995)
19. Amdahl, G.M.: Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities. In: AFIPS Conference Proceedings, pp. 483–485 (1967)
20. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A (December 2001)

Parallel Algorithms for Parabolic Problems on Graphs

Raimondas Čiegis and Natalija Tumanova

Vilnius Gediminas Technical University
Saulėtekio al. 11, LT10223 Vilnius, Lithuania
rc@vgtu.lt, tumnat@gmail.com

Abstract. In this paper two parallel numerical algorithms for solution of parabolic problems on graphs are investigated. The fully implicit and predictor-corrector finite difference schemes are proposed to approximate the differential equations on the given graph. The parallelization of the discrete algorithm is based on the domain decomposition method. Scalability analysis of the parallel algorithms is done. Some results of numerical simulations are presented and the efficiency of the proposed parallel algorithms is investigated.

Keywords: parallel algorithms, parabolic problem, graphs, finite-difference method.

1 Introduction

We consider numerical algorithms for solving linear parabolic problems on an arbitrarily branched structures. These structures can be described by using the terminology of oriented graphs (E, P), where E is a set of edges and P is a set of branching points or vertices.

Many applied problems are described by reaction - diffusion – transport equations on branched structures. Well-known examples are given by neuron simulation models which are based on the Hodgkin–Huxley reaction–diffusion system. Basic parabolic PDEs are discretized in space by using the finite-difference method. Different integration methods are used to solve the obtained systems of ODEs. Since realistic neural networks involve stiff coupled PDEs, arising from integration of individual neurons, global adaptive time-step integrators IDA and CVODES are used in [5], a local variable time-step method is investigated in [6]. Parallel versions of some algorithms, based on the domain decomposition method, are investigated in [8].

Recently, the predictor-corrector and domain decomposition methods are widely used to solve elliptic and parabolic PDEs in multidimensional domains [12]. Such algorithms are well suited for parallel implementation. Similar techniques also can be applied to solve PDE on graph structures (see [9,10] for related discussions).

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 333–342, 2012.

2 Formulation of the Discrete Model

On the graph (E, P) we consider a system of parabolic linear problems for a set of functions $\{u^k(x,t)\}$:

$$\frac{\partial u^k}{\partial t} = \frac{\partial}{\partial x}\Big(d^k \frac{\partial u^k}{\partial x}\Big) - q^k u^k + f^k, \quad 0 < x < l_k, \quad k = 1, \dots, K, \tag{1}$$
$$u^k(x,0) = u_0^k(x), \quad k = 1, \dots, K,$$

for sufficiently smooth functions d^k, q^k, f^k. At the branch points $p_j \in P$ the fluxes of the solution are conserved

$$\sum_{e_k \in N^+(p_j)} d^k \frac{\partial u^k}{\partial x}\Big|_{x=l_k} = \sum_{e_m \in N^-(p_j)} d^m \frac{\partial u^m}{\partial x}\Big|_{x=0}, \quad \forall p_j \in P, \tag{2}$$

where by $N^{\pm}(p_j)$ we denote the sets of edges, having $x = 0$ or $x = l_k$ as an end point of edge at p_j:

$$N^+(p_j) = \{e_k : \ e_k = (p_{j_s}, p_j) \in E, \ \ s = 1, \dots, S_j\},$$
$$N^-(p_j) = \{e_k : \ e_k = (p_j, p_{j_f}) \in E, \ \ f = 1, \dots, F_j\}.$$

At all vertices of the graph the continuity constraints are satisfied

$$u^m(p_j, t) = u^k(p_j, t), \quad \forall p_j \in P, \ \ e_m, e_k \in N^{\pm}(p_j).$$

A detailed description of such models is given in [1,3].

2.1 The Fully Implicit Discrete Scheme

On each edge e_k, $k = 1, \dots, K$ we define a discrete uniform spatial grid $\omega_h(k)$ and a uniform time grid ω_τ. On the grids $\omega_h(k) \times \omega_\tau$ we define discrete functions $U_j^{k,n} = U^k(x_j, t^n)$, $k = 1, \dots, K$, which approximate the solution $u^k(x_j, t^n)$ on edge e_k at time moment t^n. Differential equations (1) are approximated by the standard implicit Euler finite difference equations (see, also [3,4,7,9]):

$$U_{\bar{t}}^{k,n} = \partial_{\bar{x}}\big(d_{j+\frac{1}{2}}^{k,n} \partial_x U_j^{k,n}\big) - q_j^{k,n} U_j^{k,n} + f_j^{k,n}, \quad x_j \in \omega_h(k), \ \ k = 1, \dots, K. \tag{3}$$

The flux balance equations (2) are approximated by discrete conservation equations

$$\sum_{e_k \in N^+(p_j)} \Big[d_{N_k - \frac{1}{2}}^{k,n} \partial_{\bar{x}} U_{N_k}^{k,n} + \frac{h_k}{2}\big(U_{N_k,\bar{t}}^{k,n} + q_{N_k}^{k,n} U_{N_k}^{k,n} - f_{N_k}^{k,n}\big)\Big] \tag{4}$$
$$= \sum_{e_m \in N^-(p_j)} \Big[d_{1/2}^{m,n} \partial_x U_0^{m,n} - \frac{h_m}{2}\big(U_{0,\bar{t}}^{m,n} + q_0^{m,n} U_0^{m,n} - f_0^{m,n}\big)\Big], \ \ \forall p_j \in P.$$

At all branch points the continuity constraints are satisfied

$$U^{m,n}(p_j) = U^{k,n}(p_j), \quad \forall p_j \in P, \;\; e_m, e_k \in N^{\pm}(p_j). \tag{5}$$

The solution of finite volume scheme (3)–(4) is computed efficiently by using a modified factorization algorithm (see [3]):

- First, using the modified forward factorization algorithm, function $U^{k,n}$ is expressed in the following form

$$U_j^{k,n} = \gamma_j^{k,n} V_{k_s}^n + \alpha_j^{k,n} V_{k_f}^n + \beta_j^{k,n}, \quad k = 1, \ldots, K, \quad j = 0, \ldots, N_k, \tag{6}$$

  where V_j^n, $j = 1, \ldots, J$ are unknown values of the discrete solution at the branch points. The complexity of the factorization algorithm is $\mathcal{O}(N_1 + \ldots + N_K)$.
- Second, by substituting equations (6) into discrete flux conservation equations (4) and using the the continuity conditions (5) we obtain a system of linear equations $\mathcal{A}^n \mathbf{V}^n = \mathbf{F}^n$, where $\mathbf{V}^n = (V_1^n, \ldots, V_J^n)$ and $\mathcal{A}^n$ is a sparse matrix of dimension $J \times J$ Such systems can be solved very efficiently by direct methods, targeted for linear systems with sparse matrix, or by iterative algorithms, e.g. CG (Conjugate Gradient) type algorithms.
- During the final step, the discrete solution $U^{k,n}$ is computed using (6) and vector $\mathbf{V}^n$. The complexity of this step is again $\mathcal{O}(N_1 + \ldots + N_K)$.

Thus the total complexity of the sequential implicit algorithm is given by

$$W = W_f + W_s = \mathcal{O}\big(\sum_k N_k\big) + \mathcal{O}\big(\mu m J\big),$$

where W_f is the complexity of the factorization algorithm, and W_s is the complexity of solving the system of linear equations by using PCG (Preconditioned Conjugate Gradient) method. The dimension of this system is $J \times J$, m is the number of non-zero elements per row, and μ is the number of PCG iterations.

2.2 The Predictor-Corrector Algorithm

In this section we consider algorithms, when equations on different edges of the graph are decoupled and can be solved in parallel.

Predictor Algorithm

- First, we compute new values of the solutions at the branch points. The explicit Euler approximation is used to discretize the flux balance equations (2).
- Second, solutions at each graph edge are computed in parallel using the implicit finite difference scheme (3) with the predicted values computed at the first step as the interface boundary conditions.

Corrector Algorithm

– The main idea is to drop the values of the solution at the branch points, computed by the predictor algorithm, and to compute new values by using the basic implicit discrete algorithm.

The complexity of the sequential predictor-corrector algorithm is given by

$$W = \mathcal{O}\big(\sum_k N_k\big) + \mathcal{O}(J).$$

3 Parallel Algorithms

3.1 Parallel Fully Implicit Algorithm

Parallelization of the algorithm (3)–(4) is done by using the domain decomposition method. `Metis` tool is applied to distribute the weighted edges of the graph, where the weights are taken equal to the number of mesh points on the given edge (see, Fig. 1). Thus the accuracy of approximation and stability of the parallel algorithm coincides with the same properties of the sequential algorithm. It is well known that the convergence dynamics of the parallel iterative algorithm PCG depends slightly on the rounding errors due to floating point arithmetic and MPI data sending protocols.

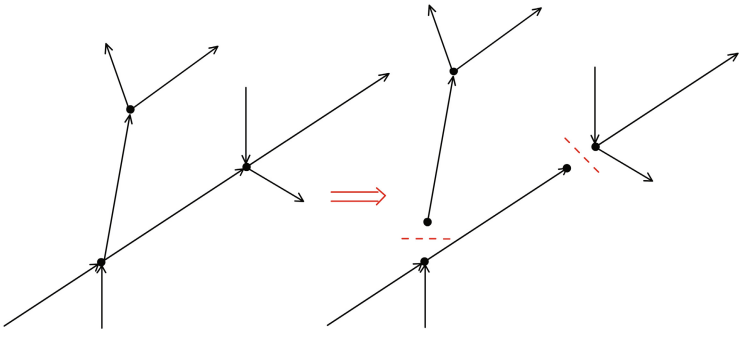


Fig. 1. Domain decomposition scheme for the implicit algorithm

Let us consider the complexity and efficiency of the parallel algorithm. Modified factorization algorithm (6) is fully parallel, and the complexity of this step is $T_{1,p} = W_f/p$, where p is the number of processes.

Now let us evaluate the complexity of PPCG (Parallel PCG) algorithm, when the diagonal preconditioner is used. After processing the forward factorization step and calculating local matrix coefficients, processors exchange data on shared vertices. The complexity of this step is $T_{2,p} = \gamma\big(\alpha + vm\beta\big)$, where $\gamma = \gamma(p)$ is the biggest number of neighbours, some process shares vertices with, v is the

biggest number of vertices, shared with the same neighbour, α is the message startup time, β is the time required to send one element of data. We use asynchronous data transfer and try to hide the costs of it by maximizing the part of computations with local data before any send or receive operation.

During matrix-vector multiplication each process exchanges ghost elements of vectors with neighbour processes. Thus the complexity of the parallel matrix-vector multiplication is given by

$$T_{3,p} = m\left\lceil \frac{J}{p} \right\rceil + \gamma\big(\alpha + v\beta\big).$$

Parallel computation of the inner product of two vectors requires global communication among all processors during summation of local parts of the product. We estimate the costs of broadcasting/reducing one item of data between p processors by $B_{1,p} = R_B(p)(\alpha + \beta)$.

When the system of linear equations is solved, the neighbour processes exchange values of solutions on neighbour vertices and the costs of this step are $\gamma\big(\alpha + v\beta\big)$.

Summing up the obtained estimates we compute the complexity of the parallel fully implicit algorithm

$$\begin{aligned} T_p = {} & \frac{W_f}{p}\delta + \frac{W_s}{p}\theta\delta + \gamma\big(\alpha + v(m+1)\beta\big) \\ & + \mu\Big(2R_B(p)(\alpha+\beta) + \gamma\big(\alpha + v\beta\big)\Big) + \gamma\big(\alpha + v\beta\big), \end{aligned}$$

where θ defines the quality of the load balancing of vertices, and $\delta = \delta(p_c)$ is the retardation coefficient, which depends on the maximum number of cores per processor. This coefficient should be taken into account, since the shared-memory structure can become a bottleneck when too many cores try to access the global memory of a node simultaneously. For the simplest data exchange algorithm and assuming that $\alpha \gg \beta$ we obtain the estimate

$$T_p \approx \frac{W_f}{p}\delta + \frac{W_s}{p}\theta\delta + \gamma\alpha\big(\mu + 2\big) + 2R_B(p)\mu\alpha. \tag{7}$$

It follows (7) that the efficiency of the implicit parallel algorithm depends on the imbalance of distributed local vertices θ. We see that the efficiency of the parallel algorithm should improve for an increased size of the problem, but the effects of worse quality of data distribution θ and negative effects of memory usage by cores can change this trend.

3.2 Parallel Predictor – Corrector Algorithm

The parallel predictor–corrector algorithm is obtained by using the domain decomposition method. In order to minimize the amount of data exchanged among processes, here we split edges, which connect vertices belonging to two different processes (see Fig. 2). On such edges, the standard factorization algorithm for

solution of systems of linear equations with three-diagonal matrices is modified by two-side version of this algorithm. Processes should exchange two factorization coefficients during implementation of this parallel version of factorization algorithm, but the complexity of the algorithm is still $8N$ floating point operations.

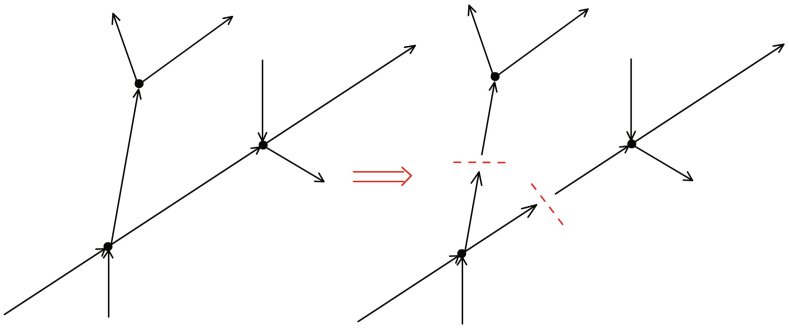


Fig. 2. Domain decomposition scheme for the predictor-corrector algorithm

The complexity of the parallel predictor-corrector algorithm is given by (for simplicity of analysis, we assume that $\theta = 1$)

$$T_p = \frac{W}{p}\delta + \gamma(\alpha + 2\beta) \approx \frac{W}{p}\delta + \gamma\alpha.$$

Since cluster communication start time α is much greater than a time required to send one number β, and messages to be sent are extremely short in our case, this evaluation can be simplified as shown. It follows from the obtained formula, that the influence of the communication costs is small, when the problem size is big enough.

The decrease in the efficiency of the parallel algorithm depends mostly on the retardation coefficient δ. After simple computations we compute the additional costs of the parallel algorithm K_p and get the following isoefficiency function W:

$$K_p(W,p) = W\,(\delta - 1) + p\gamma(p)\alpha, \quad W = \Theta\big(p\gamma(p)/(1/E_p - \delta)\big).$$

Thus, if $\gamma(p) \le \gamma_0$, then the scalability of the parallel algorithm is linear.

4 Computational Results

In this section, we present results of numerical experiments. The parabolic problem was solved on the graph presented in Fig. 3. It has 16089 edges and 15316 branching points. The lengths of the edges are distributed as $0.24 \le l_k \le 51.5, k = 1, \ldots, K$. A uniform mesh is used to approximate parabolic equations.

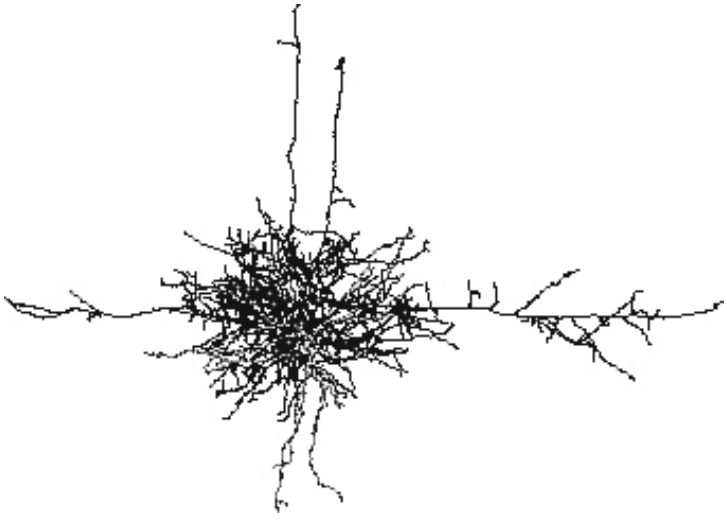


Fig. 3. The test structure of a neuron

Since the lengths of edges differ more than 200 times, balancing of sizes of local tasks becomes a very important issue. We note that the quality of partitions done by `Metis` is very close to 1.

Computations were performed on Vilkas cluster of computers at Vilnius Gediminas Technical University, consisting of nodes with Intel®Core™ processor i7-860 @ 2.80 GHz and 4 GB DDR3-1600 RAM. Each of the four cores can complete up to four full instructions simultaneously.

Obtained performance results for the parallel implicit solver are presented in Table 1. Here for each number of processors $p = n \times c$, where n denotes the number of nodes and c the number of cores per node, the coefficients of the algorithmic speed up $S_p = T_0/T_p$ and efficiency $E_p = S_p/p$ are presented. T_p denotes the CPU time required to solve the problem by using p processors. Two different space steps were used to generate the discrete spatial mesh.

In the last series of computations the influence of costs due to PPCG algorithm was investigated, therefore the number of vertices was increased twice by introducing additional vertices on each edge.

The results of computational experiments agree well with the theoretical model of the complexity of the parallel implicit algorithm. The degradation of the efficiency of the usage of cores for configurations $n \times 2$ and $n \times 4$ is obtained due to the memory bus saturation and the efficiency of cores is even decreased in a part of investigated cases as the problem size is increased. This trend is opposite to a general scalability result that the efficiency of a parallel algorithm is increasing when the size of a problem is increasing for a fixed number of processors. More results on the theoretical and computational analysis of this effect are given in [2,11].

Table 1. Computational results for the parallel implicit solver

p(nxc)	1x2	2x1	1x4	2x2	4x1	2x4	4x2	8x1	4x4	8x2	8x4
θ	1.02	1.02	1.05	1.05	1.05	1.13	1.13	1.13	1.15	1.15	1.17
$\gamma \times v$	1x5	1x5	4x4	4x4	4x4	6x4	6x4	6x4	6x4	6x4	7x3
$h = 0.03,\ J = 15316,\ T_0 = 1123$											
T_p	576	565	328	293	285	166	148	142	85	79	44
S_p	1.95	1.99	3.42	3.83	3.94	6.77	7.59	7.91	13.21	14.21	25.52
E_p	0.97	0.99	0.86	0.96	0.99	0.85	0.95	0.99	0.83	0.89	0.80
$h = 0.015,\ J = 15316,\ T_0 = 2172$											
T_p	1107	1089	620	559	546	312	284	276	161	150	83
S_p	1.96	1.99	3.50	3.89	3.98	6.96	7.65	7.87	13.49	14.48	26.17
E_p	0.98	1.00	0.88	0.97	0.99	0.87	0.96	0.98	0.84	0.91	0.82
θ	1.04	1.04	1.05	1.05	1.05	1.07	1.07	1.07	1.10	1.10	1.17
$\gamma \times v$	1x5	1x5	3x3	3x3	3x3	5x4	5x4	5x4	8x5	8x5	7x3
$h = 0.015,\ J = 30632,\ T_0 = 2250$											
T_p	1152	1133	644	588	568	331	297	286	165	157	86
S_p	1.95	1.99	3.49	3.83	3.96	6.80	7.58	7.87	13.64	14.33	26.16
E_p	0.98	0.99	0.87	0.96	0.99	0.85	0.95	0.98	0.85	0.90	0.82

In Table 2, the performance results for the parallel predictor-corrector solver are presented. We see, that the efficiency of the algorithm is very similar to the one obtained for the parallel implicit algorithm. This result is explained by the fact, that even for the parallel implicit algorithm the influence of data communication costs was small in comparison with the efficiency reduction due to the the memory bus saturation when 2 or 4 cores per node are used.

We also see some super-linear speed-up of the parallel algorithm when more nodes (not cores) are used in computations. This effect is explained by a better memory caching for smaller subproblems solved on each node. In order to test the caching effect we have solved sequential test problems of sizes N, $N/2$, $N/4$ and $N/8$. We got the following CPU times (in seconds):

$$T_1(N) = 945,\ \ T_1(N/2) = 467,\ \ T_1(N/4) = 225,\ \ T_1(N/8) = 112.$$

These results clearly show the memory caching effect.

Table 2. Computational results for the parallel predictor corrector solver

	1x2	2x1	1x4	2x2	4x1	2x4	4x2	8x1	4x4	8x2	8x4
γ	4	4	8	8	8	13	13	13	12	12	10
$h = 0.03,\ J = 15316,\ T_0 = 946$											
T_p	479	470	275	241	234	137	126	114	69	62	33
S_p	1.97	2.01	3.44	3.92	4.04	6.90	7.50	8.29	13.70	15.24	28.64
E_p	0.99	1.01	0.86	0.98	1.01	0.86	0.94	1.04	0.86	0.95	0.89
$h = 0.015,\ J = 15316,\ T_0 = 1801$											
T_p	915	898	526	459	450	263	236	224	131	122	64
S_p	1.97	2.01	3.42	3.92	4.00	6.85	7.63	8.04	13.75	14.76	28.14
E_p	0.98	1.00	0.86	0.98	1.00	0.86	0.95	1.01	0.86	0.92	0.88
γ	6	6	9	9	9	10	10	10	10	10	
$h = 0.015,\ J = 30632,\ T_0 = 1897$											
T_p	963	945	555	487	472	276	244	235	138	123	68
S_p	1.97	2.01	3.42	3.90	4.02	6.87	7.77	8.07	13.75	15.42	27.9
E_p	0.98	1.00	0.85	0.97	1.00	0.86	0.97	1.01	0.86	0.96	0.87

5 Conclusions

Two parallel numerical algorithms are developed for solution of parabolic problems on graphs. The fully implicit backward Euler and predictor-corrector finite difference schemes are proposed to approximate the differential equation. The parallelization of the discrete algorithms is based on the domain decomposition method. Results of numerical simulations show a good scalability of both algorithms. It is proved that the efficiency reduction occurs mainly due to the memory bus saturation when 2 or 4 cores per node are used.

Acknowledgment. The authors would like to thank the referees for their constructive criticism which helped to improve the clarity of this note.

References

1. Carnevale, N.T., Hines, M.L.: The NEURON Book. Cambridge University Press, Cambridge (2006)
2. Čiegis, R., Čiegis, R., Meilūnas, M., Jankevičiūtė, G., Starikovičius, V.: Parallel mumerical algorithm for optimization of electrical cables. Math. Model. Anal. 13(4), 471–482 (2008)
3. Čiegis, R., Tumanova, N.: Finite-difference schemes for parabolic problems on graphs. Lith. Math. J. 50(2), 164–178 (2010)

4. Čiegis, R., Tumanova, N.: Parallel predictor-corrector schemes for parabolic problems on graphs. Comp. Meth. Appl. Math. 10(3), 275–282 (2010)
5. Hines, M.L., Carnevale, N.T.: NEURON: a tool for neuroscientists. The Neuroscientist 7, 123–135 (2001)
6. Lytton, W., Hines, M.: Independent variable timestep integration of individual neurons for network simulations. Neural Computation 17, 903–921 (2005)
7. Mascagni, M.: The backward Euler method for numerical simulation of the Hodgkin – Huxley equations of nerve conduction. SIAM J. Numer. Anal. 27, 941–962 (1990)
8. Migliore, M., Cannia, C., Lytton, W., Markram, H., Hines, M.: Parallel network simulations with NEURON. Journal of Computational Neuroscience 21, 110–119 (2006)
9. Rempe, M.J., Chopp, D.L.: A predictor-corrector algorithm for reaction–diffusion equations associated with neural activity on branched structures. SIAM J. Sci. Comput. 28(6), 2139–2161 (2006)
10. Shi, H., Liao, H.: Unconditional stability of corrected explicit-implicit domain decomposition algorithms for parallel approximation of heat equations. SIAM J. Numer. Anal. 44(4), 1584–1611 (2006)
11. Starikovičius, V., Čiegis, R., Iliev, O.: A parallel solver for optimization of oil filters. Math. Model. Anal. 16(2), 326–342 (2011)
12. Zhuang, Y., Sun, X.H.: Stabilized explicit – implicit domain decomposition methods for the numerical solution of parabolic equations. SIAM J. Sci. Comput. 24(1), 335–358 (2002)

Efficient Isosurface Extraction Using Marching Tetrahedra and Histogram Pyramids on Multiple GPUs

Miłosz Ciżnicki, Michał Kierzynka, Krzysztof Kurowski, Bogdan Ludwiczak, Krystyna Napierała, and Jarosław Palczyński

Poznań Supercomputing and Networking Center, Poznań, Poland
{miloszc,michal.kierzynka,krzysztof.kurowski,bogdanl,
krystyna.napierala,paluch}@man.poznan.pl

Abstract. The algorithms for isosurface extraction have become crucial in petroleum industry, medicine and many other fields over the last years. Nowadays market demands engender a need for methods that not only construct accurate 3D models but also deal with the problem efficiently. Recently, a few highly optimized approaches taking advantage of modern graphics processing units (GPUs) have been published in the literature. However, despite their satisfactory speed, they all may be unsuitable in real-life applications due to limits on maximum domain size they can process. In this paper we present a novel approach to surface extraction by combining the algorithm of Marching Tetrahedra with the idea of Histogram Pyramids. Our GPU-based application can process CT and MRI scan data. Thanks to domain decomposition, the only limiting factor for the size of input instance is the amount of memory needed to store the resulting model. The solution is also immensely fast achieving up to 107-fold speedup comparing to a serial CPU code. Moreover, multiple GPUs support makes it very scalable. Provided tool enables the user to visualize generated model and to modify it in an interactive manner.

Keywords: isosurface extraction, marching tetrahedra, histogram pyramids, multiple GPUs applications.

1 Introduction

Surface reconstruction from volumetric data is used to visualize and analyze the output acquired from different types of material scanners, such as Magnetic Resonance Imaging (MRI), Computed Aided Tomography (CT) or Positron Emission Tomography (PET). Analysis of volumetric data is essential in many domains, among which are medical applications, geoscience and computational geometry. In petroleum engineering, Computed Tomography is used as a non-destructive imaging method producing representation of the internal structure of pores in reservoir rocks. Resulting 2D images are processed to generate 3D models of the rock structure, which can be further used in reservoir simulations to estimate the recovery factor. In medical imaging, creating a constant density 3D surface

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 343–352, 2012.

from 2D slice images enables to extract and visualize separately different kinds of tissues, such as bones, soft tissues or muscles. 3D display gives an additional insight into medical data and helps to make an accurate assessment of the patient. Isosurface extraction is also used in visualizations where the data itself varies over time, e.g. in CFD simulations.

Surface reconstruction is a very computationally intensive process with high memory requirements, as n^3 vertices have to be analyzed for the data resolution of n. Moreover, the model is created for a given density at a time, so to build a surface at a different density, the computations have to be repeated. The implementations on standard serial devices such as CPUs are inefficient for the demands of today's market. In medical applications, the possibility of analyzing the data interactively in the real time is crucial for the fast diagnosis and treatment. In petroleum engineering, huge volumes of data produced from rock scans have to be analyzed in a reasonable time. The implementations on standard serial devices such as CPUs cannot meet such requirements. Therefore, the research concentrates nowadays on highly parallel and power-efficient devices. Since the problem of surface construction consists in performing intensive local computations, it may be executed very efficiently on graphics cards, which offer high memory bandwidth and hundreds of computational units capable of performing massive local computations in parallel.

In this paper we present an efficient, GPU-based implementation of Marching Tetrahedra – an algorithm for isosurface extraction that runs entirely on modern graphics cards and as such is immensely fast. Memory complexity problem is addressed by the algorithm of Histogram Pyramids [1,2]. In the experimental study we show that our implementation outperforms serial code running on CPU. Moreover, multiple GPUs support makes it very scalable, enabling to create robust models for large amounts of data in a satisfactory time. Finally, we provide the user with an application for viewing and manipulating generated model.

2 Background

2.1 Isosurface Extraction Methods

There are several methods of extracting the isosurface from volumetric data. One of the most popular is Marching Cubes (MC) [3]. The generated surface separates voxels with value greater than V from the other voxels. The value of a voxel usually describes the density of a material at a given point. MC divides the input domain into cubes (hexahedrons) with input data voxels on the cubes' vertices. Then, locally for every cube, it creates a part of the isosurface. Each vertex of a cube can be in one of two states ($v_i > V$ or $v_i \leq V$), hence, the isosurface can intersect every cube in one of the $2^8 = 256$ cases. However, the method has a few drawbacks. First, its table of cases is ambiguous and as a result special cases have to be considered. Nevertheless, some incorrect connections may be generated anyway [4]. Secondly, the amount of memory needed for so called lookup table makes it very hard to implement properly on a GPU.

Another method for extracting the isosurface is Marching Tetrahedra (MT). Its main idea is derived from MC, however, the difference lies in the division of the volumetric data. Instead of creating virtual cubes, MT creates virtual tetrahedra. As tetrahedron has only 4 vertices, the isosurface can intersect it in only $2^4 = 16$ different cases. MT eliminates the ambiguity of MC without adding any special conditions. This property is particularly desirable for the GPU architecture where the code is executed much more effectively if there are no conditional statements. Furthermore, MT's tiny lookup table fits in with the GPU architecture much better allowing for better utilization of its parallel units.

In both methods, the upper bound of memory needed to store all the potential vertices of the extracted isosurface is much higher than the memory needed for vertices actually produced. This is a substantial problem for GPU computing in which dynamic allocation of memory is hardly possible. In [2] the authors showed that incorporating the algorithm of Histogram Pyramids [1] into GPU implementation of MC may be very beneficial since the method allows to compute the total amount of memory needed as well as facilitates parallel access to the memory. Because MC and MT are analogous as it comes to memory access patterns, we have decided to involve the idea of Histogram Pyramids into our algorithm too.

2.2 Related Works

Ever since the computational power of graphics cards started to exceed the capabilities of traditional CPUs, scientists around the world have been taking advantage of this fact. As a result, there are a number of GPU-accelerated applications nowadays, including solutions addressing isosurface extraction problem. Historically, the first approaches to the Marching Tetrahedra algorithm on GPU used OpenGL and graphics pipeline to generate the isosurface. Pascucci [5] presented an implementation based on a table of edges and a table of cases. Similar method was also used by Reck et al. [6]. It should be stressed, however, that in both cases the computed mesh had to be rendered directly and therefore could not be reused, e.g. in the next frame. Klein et al. [7] evade this limit by using Open GL SuperBuffer objects to store the result of the surface extraction. The method, however, was able to handle relatively small instances only due to its high memory consumption, which in turn was slightly enhanced by Kipfer et al. [8]. Buatois et al. [9] introduced a few improvements to [5] and, still using vertex shader units, was able to process bigger instances (up to 5 million tetrahedra) obtaining speedup of around 2 in comparison to serial CPU code. Yet, with the advent of new graphics hardware came new possibilities. The application presented by Tatarchuk et al. [10] certainly may be considered as state-of-the-art in this area. It combines MC and MT to perform fast and accurate isosurface extraction using geometry shader units available on DirectX 10 compliant GPUs. The solution, tested on ATI Radeon 4870, reached up to 24 million faces generated per second. Moreover, it allows the user to change the isovalue on demand. However, its drawback lies again in memory consumption. Maximum grid size for which the authors presented results was only 64^3. There

are also a few solutions that focus mainly on volume rendering [11,12], and as such do not place too much emphasis on isosurface extraction. Additionally, it is worth noting that none of the cited works utilizes more than one GPU. Bearing in mind all the limitations of previous methods, we set out to design and implement an algorithm that could efficiently extract isosurface, especially for large datasets, using a single as well as multiple GPUs systems.

3 Marching Tetrahedra and the Idea of Histogram Pyramids

The input data for the algorithm of Marching Tetrahedra is given as a regular cubic 3D grid. Each vertex in the grid defines a 3D position with x, y, z coordinates and a density value. MT constructs a set of triangles connecting the vertices with a specific density value. These planar triangles approximate the isosurface (surface constructed for a given density, called isovalue) to be visualized. The finer the grid, the closer the approximation to the actual surface, resulting in a more accurate model. However, a finer grid produces more triangles and hence memory and processing time requirements grow.

Marching Tetrahedra constructs the isosurface by splitting each grid cell into 6 tetrahedra. Although it would be possible to split the grid cell into 5 tetrahedra (see [13]), we consider only the traditional approach in this paper. A tetrahedron is defined by the values at its four vertices. Each vertex is classified as either above or below given isovalue. If some vertices of a tetrahedron are below the isovalue and some are greater than this value, the tetrahedron contributes to the isosurface and eight different cases can be distinguished (see Figure 1), otherwise the tetrahedron has no influence on the resulting isosurface. An edge between two adjacent vertices is intersected by the isosurface if one vertex is above the isovalue and the other is below. A position of the intersection is computed by linear interpolation - proportionally to the ratio of the values at the vertices.

At the beginning of the algorithm it is not known whether given tetrahedra will contribute to the resulting mesh or not. Hence, the amount of memory to allocate is also unknown. Another problem arises when the tetrahedra are processed simultaneously (e.g. on a GPU) and the results must be written to the contiguous memory in parallel. It is possible to append results to the memory by using an atomic address counter. Such solution, however, would decrease the level of parallelism and ruin the performance (cf. Section 5). To address these memory related problems we incorporated the idea of Histogram Pyramids into the algorithm of MT.

The Histogram Pyramid is used to create the output stream of vertices (forming the triangles of the isosurface) from the input stream of tetrahedra contributing to these vertices (cf. [2]). The idea of 1D Histogram Pyramid is presented in Figure 2. The bottom of the pyramid is called the base layer, and contains the sequence of input elements. Each input element corresponds to a single tetrahedron and contains the information about the number of triangles that are to be produced by this tetrahedron. From the input layer, the Histogram Pyramid

is built bottom-up, layer by layer. Each consecutive layer is half the size of its predecessor. An element in a given layer is the sum of the two corresponding elements from the layer below. Since there are no dependencies between elements in the same layer, they can be calculated in parallel.

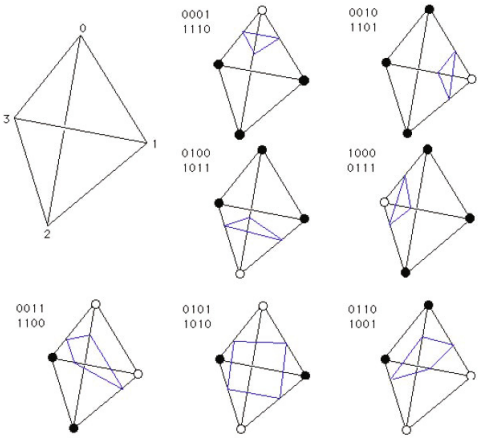


Fig. 1. Eight cases of tetrahedra. The figure comes from the website http://paulbourke.net/geometry/polygonise/#tetra.

Afterward, the Histogram Pyramid is used to produce the output stream, i.e. triangles comprising the surface. Since the last layer with only one element refers to the total number of the triangles in the model, it is used to determine the amount of memory needed. For each tetrahedron considered in this phase, the Histogram Pyramid is traversed top-down to find the right place in the memory for the output triangle. This eliminates the necessity of maintaining the atomic counter of memory address.

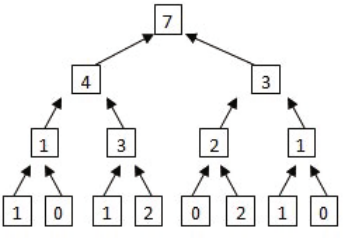


Fig. 2. Histogram Pyramid construction

4 Implementation of the Algorithm

Efficient implementation on a GPU is more demanding than standard CPU programming. First, several kinds of memory with different capacities, characteristics and access times need to be considered to choose a proper one for a given task. Second, calculations are the most efficient when every thread performs the same series of operations. Although conditional statements are allowed, they significantly decrease the performance [14]. The latter was one of the reasons why the Marching Tetraheda algorithm was chosen to be implemented. We decided to develop our application using CUDA library.

Our first approach to the problem works as follows. The input data is stored in the *global memory* as a 3D texture (cf. [14]). The domain is divided into blocks of threads. Each block contains $t_x \times t_y \times t_z$ threads. Each thread from the block is responsible for copying one corresponding voxel from the input domain to the *shared memory*. The next stage is performed only by $(t_x - 1) \times (t_y - 1) \times (t_z - 1)$ threads, since this is the number of cubes that may be constructed from fetched data. Each thread divides the corresponding cube into six tetrahedra and creates triangles according to MT. However, the main drawback of such straightforward approach (without Histogram Pyramids) is unpredictable amount of memory needed to store output vertices and lack of information which voxels actually produce isosurface triangles. To round this problem, the program initially allocated relatively large amount of memory (chosen beforehand) to provide storage for all possible triangles. To assure the continuity of output vertex stream among different threads, an atomic counter was introduced. The counter, incremented every time a thread created a triangle, was treated as a memory offset. This solution, however, turned out to be significantly slow. Therefore, we decided to make use of the Histogram Pyramids method (HP). From this point onwards, our algorithm consists of two phases: Histogram Pyramid construction and computation of the polygonal mesh.

Our implementation uses 1D version of HP, since we found one dimensional arrays as the best to manage on a GPU. The input data is represented in four-dimensional space, where the first three dimensions refer to the position of a cube in volumetric data and the fourth one identifies a tetrahedron within a given cube. Hence, the position p in 1D HP input stream is calculated according to the following formula:

$$p = (((x \cdot y_{dim} + y) \cdot z_{dim} + z) \cdot w_{dim}) + w \tag{1}$$

where x, y, z indicate the position of a cube in the volumetric data, x_{dim}, y_{dim}, z_{dim} are corresponding sizes of the domain, w identifies tetrahedron within a given cube and w_{dim} is the size of this dimension ($w_{dim} = 6$).

In order to construct the HP, the input domain is scanned by blocks of threads in search for tetrahedra that will contribute to the output mesh. This time, a single thread is launched for each tetrahedron. During this step the input stream

of HP is filled with information determining the number of triangles to be created by the corresponding tetrahedra, i.e. with integer numbers between 0 and 2. It is worth noting that in order to save memory this information is stored using only two bits. Subsequently, each level of the HP is computed in parallel. When the top level of the HP is reached, the amount of memory needed to store the isosurface is known. The memory is allocated and the algorithm proceeds to the second phase.

The second phase is responsible for actual computation of the isosurface. The program launches exactly one GPU thread for each triangle to be created. The information about the number of threads is read from the top level of HP. Each thread fetches the input voxels (based on its position p) and calculates the triangle correspondingly. Next, it stores the results at its exclusive memory address which is computed from the HP by a simple process of traversing its structure. The advantage of this approach is that all the threads may execute their operations independently (HP is read-only here) and this perfectly fits in with the GPU architecture.

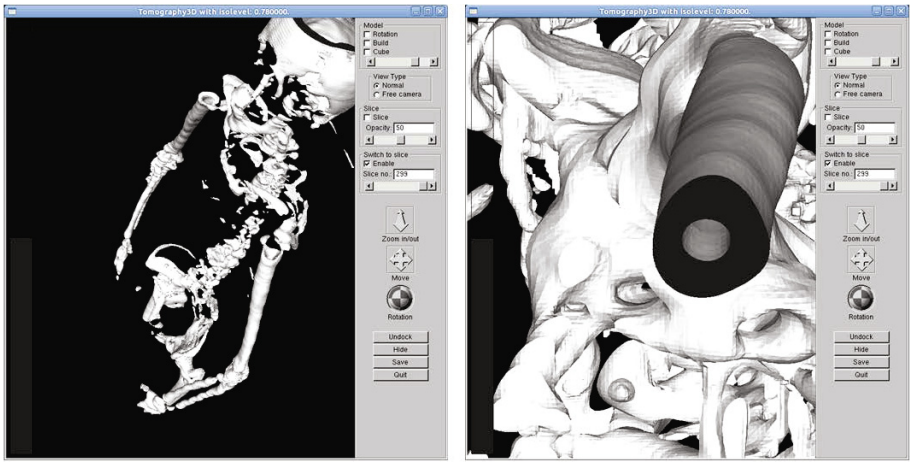


Fig. 3. Two exemplary screenshots of the application. The domain size in both cases was $512 \times 512 \times 300$. The panel on the right enables the user to manipulate the model interactively, e.g. by changing the isovalue.

Bearing in mind that nowadays many workstations are equipped with more than one GPU, we also designed and implemented multiple GPUs support. Its main idea is that the input domain is not sent entirely to a single GPU, but instead is divided into subdomains, each of which is processed independently. Theoretically, each subdomain could be processed on a separated GPU, but in practice, for large models, the number of subdomains is greater than the number of GPUs available in the system. Hence, we implemented also a simple load balancing mechanism. Each GPU that becomes idle receives a new portion of

domain to compute. Its size depends on total amount of *global memory* available on a given GPU. As parts of the model are generated on different GPUs, the final model is merged in the main host memory (RAM), which, in fact, is the only limiting factor for domain size that can be processed. If necessary, the final model is send back to a graphics card that is in charge of displaying. On the other hand, for small enough models that fit into single GPU memory, to avoid costly data transfers through PCI-E bus, the model is entirely computed on a single GPU and directly displayed using OpenGL Vertex Buffer Object. Exemplary screenshots of the application are presented in Figure 3.

5 Results

5.1 Time Comparison to the Serial Implementation

In order to compare the performance of parallel and sequential implementations, three versions of the algorithm were used: CPU-based sequential version of MT (CPU-MT), GPU version of MT without HP (GPU-MT) and with HP (GPU-MT-HP). Computations were performed on Intel Core 2 Quad 2.0GHz CPU with a single NVIDIA GeForce GTX 285 graphics card. Tests were repeated 10 times and the average values are presented. For testing purposes ,,MRI of the Skull Base" [1] dataset with different resolutions and different numbers of slices was used. Results are presented in Table 1 and in Figure 4.

The results show that both GPU implementations of the MT method outperform its CPU equivalent, reaching the speedup of up to 107-fold (GPU-MT-HP). While we are aware that our CPU implementation may be not optimal, the measurements show the following general trend: the bigger the model the greater speedup is achieved. Our algorithm generated here up to 28 million faces per second which is slightly more than 24 million achieved by Tatarchuk et al. [10]. It is also clear that the algorithm using Histogram Pyramids is much more efficient. Moreover, Figure 4 shows that the time needed by GPU-MT-HP is more predictable. Since GPU-MT-HP requires less memory, we may conclude that it dominates GPU-MT.

Table 1. Mean computational times (in seconds) of three implementations of the MT algorithm for different domain sizes. The ,,speedup" rows were computed with CPU-MT times as a reference.

	34^2 x 5	102^2 x 15	170^2 x 25	238^2 x 35	307^2 x 45	375^2 x 55	443^2 x 65	512^2 x 75
CPU-MT	0,034	0,162	0,755	1,759	3,594	6,204	9,728	14,391
GPU-MT	0,004	0,019	0,063	0,098	0,225	0,300	0,370	0,410
speedup	9	9	12	18	16	21	26	35
GPU-MT-HP	0,003	0,003	0,008	0,017	0,036	0,058	0,092	0,135
speedup	11	54	94	103	100	107	106	107

[1] `http://www.mr-tip.com/serv1.php?type=slider&slide=MRI` of the Skull Base

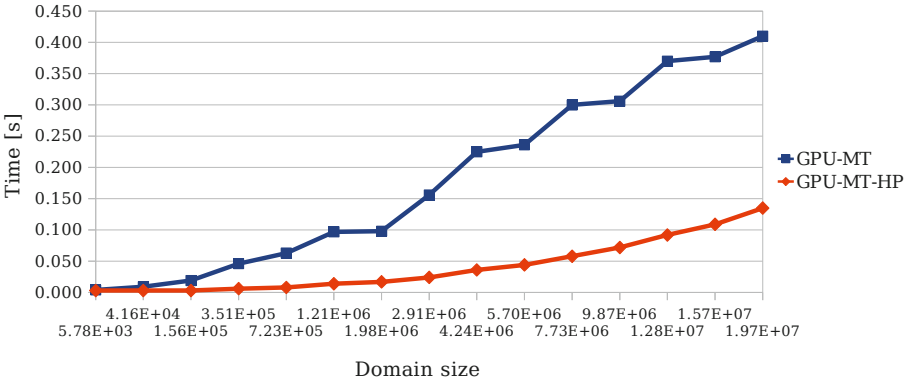


Fig. 4. The impact of Histogram Pyramids on computational time[s] of the GPU-based MT algorithm

5.2 Multi-GPU Test

The multi-GPU test was performed to see how the performance of the algorithm depends on the number of graphics cards used. The GPU-MT-HP algorithm was run on a computer equipped with NVIDIA Tesla S1070, i.e. 4 × GPU with 4GB of RAM, 960 cores in total. Time measurement included: copying input data to GPUs memory, model computation and transfer of results to back the host memory. The domain size was constant (512 × 512 × 300). Tests were repeated 10 times and the average speedups in comparison to a single GPU were: 1.9, 2.6 and 3.3 for two, three and four GPUs, respectively. Since the jobs that are processed on separated GPUs are independent (no communication needed), one could expect a perfectly linear speedup. We have thoroughly checked the influence of load balancing which is, however, negligible here. The observed decrease in performance is caused instead by a limited bandwidth of host memory as each GPU requires transfers of a notable amount of data. Nevertheless, the test shows that our implementation certainly benefits from multiple GPUs systems.

6 Conclusions

The paper presents an efficient parallel implementation of the Marching Tetrahedra algorithm for isosurface extraction. The method, optimized by using Histogram Pyramids, runs entirely on graphics cards and as such is immensely fast. Moreover, multiple GPUs support make the application very scalable. As a result, it allows to analyze and visualize larger data sets, e.g. CT images representing larger rock formations for reservoir simulations in oil&gas industry. In medical imaging, faster computations allow the physicians to analyze the 3D model of a patient at several density values. Furthermore, provided tool enables the user to interactively manipulate on generated model, e.g. by switching

between isovalues almost in real-time. Since one of our priority is to process real-life rock formations, our future development will concentrate mainly on further parallelization (CUDA+MPI) of presented method.

Acknowledgments. The presented work was sponsored by the UCoMS project under award number MNiSW (Polish Ministry of Science and Higher Education) Nr 469/1/N – USA/2009 in close collaboration with U.S. research institutions involved in the U.S. Department of Energy (DOE) funded grant under award number DE-FG02-04ER46136 and the Board of Regents, State of Louisiana, under contract no. DOE/LEQSF(2004-07).

References

1. Ziegler, G., Tevs, A., Theobalt, C., Seidel, H.P.: GPU Point List Generation through Histogram Pyramids. Technical report MPI-I-2006-4-002, Max-Planck-Institut fur Informatik (2006)
2. Dyken, C., Ziegler, G., Theobalt, C., Seidel, H.P.: High-speed Marching Cubes using Histogram Pyramids. Computer Graphics Forum 27(8) (2008)
3. Lorensen, W.E., Cline, H.E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. SIGGRAPH Computer Graphics 21(4), 163–169 (1987)
4. Ning, P., Bloomenthal, J.: An Evaluation of Implicit Surface Tilers. IEEE Computer Graphics and Applications 13(6), 33–41 (1993)
5. Pascucci, V.: Isosurface Computation Made Simple: Hardware Acceleration, Adaptive Refinement and Tetrahedral Stripping. In: IEEE TVCG Symposium on Visualization 2004, pp. 293–300 (2004)
6. Reck, F., Dachsbacher, C., Grosso, R., Greiner, G., Stamminger, M.: Realtime Isosurface Extraction with Graphics Hardware. In: Proceedings Eurographics (2004)
7. Klein, T., Stegmaier, S., Ertl, T.: Hardware-accelerated Reconstruction of Polygonal Isosurface Representations on Unstructured Grids. In: Proceedings of Pacific Graphics, pp. 186–195 (2004)
8. Kipfer, P., Westermann, R.: GPU Construction and Transparent Rendering of Iso-Surfaces. In: Proceedings Vision, Modeling and Visualization 2005, pp. 241–248. IOS Press, infix (2005)
9. Buatois, L., Caumon, G., Lvy, B.: GPU Accelerated Isosurface Extraction on Tetrahedral Grids. In: International Symposium on Visual Computing (2006)
10. Tatarchuk, N., Shopf, J., DeCoro, C.: Advanced interactive medical visualization on the GPU. Journal of Parallel and Distributed Computing 68(10), 1319–1328 (2008)
11. Maximo, A., Marroquim, R., Farias, R.: Hardware-Assisted Projected Tetrahedra. Computer Graphics Forum 29(3), 903–912 (2010)
12. Kalbe, T., Koch, T., Goesele, M.: High-Quality Rendering of Varying Isosurfaces with Cubic Trivariate C1-continuous Splines. In: Proceedings of 5th International Symposium on Visual Computing (2009)
13. Zilinskas, A., Zilinskas, J.: Global Optimization Based on a Statistical Model and Simplicial Partitioning. Computers & Mathematics with Applications 44(7), 957–967 (2002)
14. NVIDIA CUDA Best Practices Guide 3.2 (2010)

Parallel Implementation of Stochastic Inversion of Seismic Tomography Data

Maciej Dwornik and Anna Pięta

AGH University of Science and Technology,
Faculty of Geology, Geophysics and Environment Protection,
Department of Geoinformatics and Applied Computer Science,
al. A. Mickiewicza 30, 30-059 Kraków, Poland
maciek.dwornik@wp.pl, apieta@geol.agh.edu.pl

Abstract. In this paper parallel implementation of stochastic inversion of seismic tomography data was presented. Classical approach to travel time tomography assumes straight line of seismic rays between sources and receives points and isotropy of geological medium. Such simplifications are potential sources of inaccuracy of the obtained results of travel time tomography. Stochastic methods can be free from these simplifications. On the other hand, this kind of algorithms requires huge number of time consuming calculations. In this work parallelization of two stochastic methods is presented. Different parallel algorithms was applied to presented inversion problems. Obtained results show significant differences in parallel performance of presented inversion algorithms.

Keywords: seismic tomography, inverse problem, seismic anisotropy, master-slave paradigm.

1 Introduction

Seismic travel time tomography is one of the most powerful tools to obtain distribution of elastic parameters in geological medium. Precise information about these parameters can be helpful, especially in solving environmental engineering problems, oil and gas exploitation or evaluations safety in mines. Traditional tomography method commonly assumes simplified geological medium without differences of velocity of seismic wave propagation in different direction and straight line seismic rays between source and receive points. These assumptions are necessary for linearization of inversion process. Stochastic methods are used for solving inverse problems, where such assumptions are not necessary. Stochastic methods, such as Monte Carlo method or genetic algorithms presented in this paper, are examples of very time consuming algorithms. Parallelization of the calculations is the only method, that can decrease time of calculation without decreasing accuracy of the obtained results. This method is commonly used for solving geophysical modeling and inversion problems (f.e. [2], [1]).

In this paper application of parallel computing of the inversion of the seismic tomography data is presented. The most time-consuming part of the inversion

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 353–360, 2012.

in both presented methods is the modeling of the seismic wave propagations. It is done for every data set in every iteration of the algorithm. High accuracy of the obtained results requires huge amount of iterations.

2 Description of the Seismic Tomography Inversion Algorithm

Inversion of seismic tomography data is a process, where velocity field is obtained using observed travel times of wave's propagation. It is strongly non-uniqueness problem. For the simplest scheme of inversion velocity field can be estimate using matrix decomposition or algebraic reconstruction technique [3]. In case of anisotropy and wave propagation in real geological medium, calculation of velocity field is extremely difficult. Stochastic type of inversion is done in this case by evaluation of randomly chosen velocity model. It cause that the most time consuming part of seismic inversion is calculation of seismic wave propagation, so called forward problem.

Seismic anisotropy is phenomena of different velocities of wave propagation in different directions. Several models of anisotropy exists: from the simplest one, based on eclipse (ellipsoid in 3D), through transversally isotropy (VTI, HTI, TTI), to the most general case, described by 21 coefficients of stiffness tensor. In this work VTI (vertical transversally isotropy) model was ssumed. It is describe by five stiffness tensor coefficients and density. This model describes seismic wave propagation in finely laminated, horizontal layered medium (f.e. shales).

Forward problem was solved using the shortest path method. This method is fast and stable and gives precise results for random velocity field. It is based on Fermat's law and Huygens' principles. Description of this algorithm could be find in [10], [5], [4].

Two stochastic methods of inversion were used to estimate values of stiffness tensor coefficients and density: Monte Carlo method with and without local searching and genetic algorithm. Objective function must minimize error function given by equation:

$$f(C) = \|t^{obs}, t^{est}(C, G)\| \tag{1}$$

where C – two dimensional distribution of stiffness tensor value,
G – geometry of acquisition (location of shooting and receive points)
t^{obs} – observed travel time of primary seismic wave.

Error function have at least one global minimum equal to zero, because inverse problem is always non-uniqueness, This function calculate error of reconstruction based on comparison between observed data (theoretical travel times) and estimated from stiffness tensor distributions. Two norms of calculation error function were used: mean of absolute values (L1) and RMS (L2) between both time vectors. Ranking list of obtained solutions with the smallest errors was updated in each iteration.

2.1 Monte Carlo Method

Monte Carlo is classical algorithm, where evaluation is done for the value of parameters chosen randomly from bounded space. This algorithm is widely used for solving geophysical inverse problem (f.e.[9]). It assumed, that only data sets with the smallest error function are saved. Stop condition for Monte Carlo algorithm was set, when maximum number of iteration was reached or the number of tested data sets with error smaller than assumed was obtained. This algorithm was enhanced by local searching operator. Local searching operator shrink searching space to subspace near considered data set, if there was change on ranking list in previous iteration.

2.2 Genetic Algorithms

Genetic algorithms are global optimization method, which simulate evolution process [8]. This method is commonly used in many seismic applications, f.e. in ray tracing [11], inversion [6]. In this inversion floating coding was used for code individuals – in this case, parameters of seismic velocity field. Main loop is organized as follow (description after):

```
while condition is true
  add new to population (M)
  crossover (M)
  mutation (S)
  evaluate (S)
  ranking list (M)
  evolution (M)
end
```

(Main computational loop of genetic inversion of velocity of anisotropic medium. Letters in blanket refer to the parallel algorithm and indicate which operations are done by master node (M) and which can be done simultaneously (S).)

Main loop is executed until the condition evaluates to false. We assumed the same stop condition for both Monte Carlo and genetic algorithms. Six various operations on analyzed data are done in each iteration:

- changing value of individual's component (stiffness tensor coefficient) with assumed probability, that changes given value by adding randomly value from normal distribution with assumed variation and zero mean (mutation);
- linear combination of the components of the two different data vector coefficients (crossover);
- evaluation of each individual using norm L1 or L2 separately, which implies two separate calculations for different populations;
- evolution by tournament mode, in which individuals of new population are obtained from individual chosen randomly from old population;
- changing the worst individual by randomly chosen generated;
- ranking list, which guarantees surviving of the best individual even if it do not pass to the next population during tournament mode.

2.3 Example Results

The aim of the inverse algorithm was to obtain the velocity of primary wave in horizontal layered geological medium built with three layers with vertical fault. The height of the model was Z=150m and its width X was equal to 100m. It was comprise of 150 velocity cells. Velocity of primary wave in vertical transversely isotropy medium (VTI) is given by follow equation [13]:

$$v_P^2(\theta) = \frac{1}{2\varrho}\left[c_{33} + c_{44} + (c_{11} - c_{33})\sin^2\theta + D(\theta)\right]$$
$$D(\theta) = \{(c_{33} - c_{44})^2 + 2\left[2(c_{13} + c_{44})^2 - (c_{33} - c_{44})(c_{11} + c_{33} - 2c_{44})\right] \cdot$$
$$\cdot \sin^2\theta + \left[(c_{11} + c_{33} - 2c_{44})^2 - 4(c_{13} + c_{44})^2\right]\sin^4\theta\}^{0.5} \quad (2)$$

It caused that for each velocity cell four coefficients of stiffness tensor (c_{11}, c_{13}, c_{33} and c_{44}) and density have to be estimated. It gave 750 values of independent variable to obtain. Parameters of the geological medium, that should be obtained in inversion process, are presented in Table 1.

Table 1. Value of elastic parameters used for modeling and inversion

Parameter	Layer 1	Layer 2	Layer 3
Type	Isotropy	Anisotropy	Isotropy
c_{11}[GPa]	8.0	16.6	20.0
c_{13}[GPa]	2.0	6.5	4.0
c_{33}[GPa]	8.0	10.3	20.0
c_{44}[GPa]	3.0	3.0	8.0
Density [kg/m^3]	2000	2000	2000

As it was shown on the picture below (Fig.1), if the number of iterations of inversion procedure increases, differences between travel times, obtained from estimated anisotropy parameters and values received from inversion procedure, significantly decrease. Thus more iteration we perform, better results of stochastic inversion we receive. The results obtained from inversion process are satisfactory only for GAs. In this case one million iterations is not enough for obtain acceptable solution using MC method.

Because average time of single evaluation is about 3 seconds, it is almost impossible to estimate velocity field using single computer.

3 Parallel Implementation and Discussion

Many parallelization strategies exists for optimization problems [12]. In this work two different parallel algorithms were developed to utilize parallel computational environment in inversion of density and anisotropy components of the given geological model. In Monte Carlo inversion independent calculation of velocity field

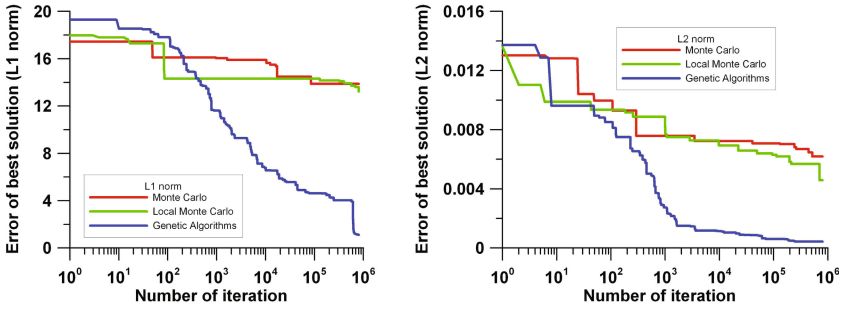


Fig. 1. Relationship between error of the best solution obtained during inversion procedure and the number of iteration (left - L1 norm, right - L2 norm). Number of iteration in genetic algorithms mean number of evalueted individuals (number of population multiply number of individuals)

for a given parameter set were performed on different processes. Results of inversion were after sent to a chosen computational process. In genetic algorithm, parallelization was done by distribution of the population among all available processes in each iteration. For such divided subpopulations the most computational expensive operations such as evaluation of the objective function and mutation were performed by all processes separately. Computational results were then sent to master process where ranking list together with other operations listed before were preformed. The new population created in evolution stage were then split and sent to computational processes. Operations described before were performed until desire accuracy or assumed maximal amount of iterations was reached.

For both algorithms statically task distribution was assumed. Static scheduling for fine grain Monte Carlo inversion minimizes communication delay. Computational time for single objective function evaluation has almost the same values so static task distribution maximizing resource utilization for both inversion algorithms. Schemas of parallel algorithms are presented below (Fig.2).

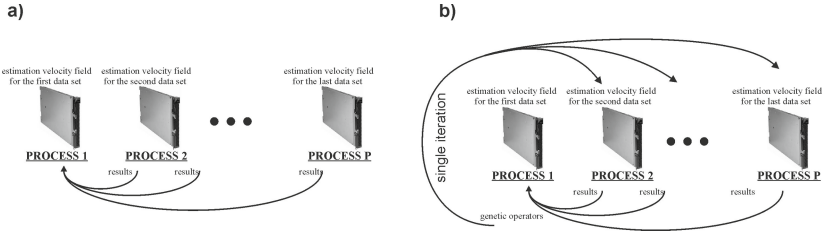


Fig. 2. Schema of parallel inversion of velocity model for Monte Carlo (a) and genetic algorithm (b)

For both parallel algorithms commonly used parameters such as computational time, speedup and efficiency that describe parallel performance of the algorithm was computed. Parallelization of Monte Carlo and genetic algorithms, using MPI [7] library was tested on of IBM Blade machine with 8 computational nodes 2GHz and 4 GB RAM each. Inversion was performed for the model described in previous chapter. Both algorithm performance similar number of computation of velocity field: 5000 iterations of Monte Carlo algorithm were assumed whereas for genetic algorithm velocity computation was done for 48 individuals in 100 iterations.

Monte Carlo inversion is an example of embarrassingly parallel algorithm, where many independent tasks are solving simultaneously and there is no need for coordination among task. This algorithm is fully scalable, its speedup increases linear (Fig.3) and its efficiency is almost 100% (Fig.4). Fluctuation in efficiency curves is mainly due to no equal dividing of computational problem among processes. It can be also caused by slightly differences of computational time of the single iteration of forward problem. Parallel genetic algorithm does

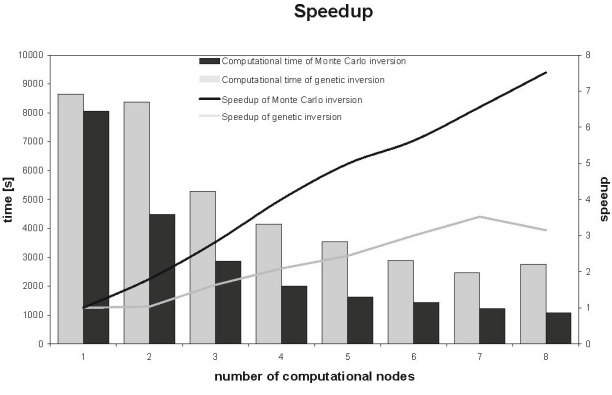


Fig. 3. Computational time and speedup for Monte Carlo and genetic inversion algorithm

not present such excellent performances as Monte Carlo inversion. In this case master slave paradigm was applied because of genetic operators which require sequential processing. The need for sequential processing in every iteration of genetic algorithm effect frequent synchronization and communication during calculation. Moreover in this algorithm huge size of data sets has to be send before each computational step which additionally decreases parallel performance of genetic algorithm (see Fig.4) and decrease of efficiency among one and two computational processes).

Two different algorithms were applied for the problem of inversion of velocity field. Parallel computing represents a natural paradigm for Monte Carlo methods. Analyses of parallel performance done for this algorithm show almost 100% utilization of computational resources. For communication intensive applications, such as genetic algorithm computational resources are not fully used.

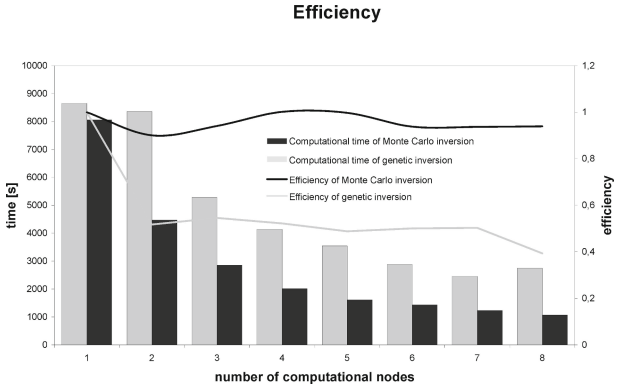


Fig. 4. Computational time and efficiency for Monte Carlo and genetic inversion algorithm

The bottleneck of the genetic algorithm is existence of sequential code sections and frequent communication that regard huge data sets. Modifications of message size or frequency of communication may improve parallel performance for genetic algorithm.

Acknowledgements. The study was financed from the statutory research project No 11.11.140.032 of the Department of Geoinformatics and Applied Computer Science, AGH University of Science and Technology.

References

1. Bała, J., Pięta, A.: Comparing parallel programming environments for the joint inversion of geoelectrical data. Computer Science 10, 85–95 (2009)
2. Danek, T., Franczyk, A.: Parallel and distributed seismic wave-field modeling. TASK Quarterly: Scientific Bulletin of Academic Computer Centre in Gdansk 8(4), 573–582 (2004)
3. Dwornik, M., Leśniak, A.: Comparison between Inversion Algorithms of Surface Refraction Tomography for Cavities Detection. In: 13th European Meeting of Environmental and Engineering Geophysics, Istanbul, p. 22 (2007)
4. Dwornik, M., Pięta, A.: Efficient Algorithm for 3D Ray Tracing in 3D Anisotropic Medium. In: 71st EAGE Conference & Exhibition incorporating SPE EUROPEC 2009, Extended Abstracts (2009)
5. Fischer, R., Lees, J.L.: Shortest path ray tracing with sparse graph. Geophysics 58, 987–996 (1993)
6. Gerstoft, P.: Inversion of seismoacoustic data using genetic algorithms and a posteriori probability distributions. J. Acoust. Soc. Am. 95, 770–782 (1994)
7. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface. MIT Press, Cambridge (1994)
8. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer (1998)

9. Mosegaard, K., Sambridge, M.: Monte Carlo analysis of inverse problems. Inverse Problems 18, R29–R54 (2002)
10. Moser, T.J.: Shortest path calculation of seismic rays. Geophysics 56, 59–67 (1991)
11. Sadeghi, H., Suzuki, S., Takenaka, H.: A two-point, three-dimensional seismic ray tracing using genetic algorithms. Physics of the Earth and Planetary Interiors 113, 355–365 (1999)
12. Schnabel, R.B.: A View of the Limitations, Opportunities and Challenges in Parallel Nonlinear Optimization. Parallel Computing 21, 875–905 (1995)
13. Thomsen, L.: Weak elastic anisotropy. Geophysics 51, 1954–1966 (1986)

Parallel Coarse-Grid Treatment in AMG for Coupled Systems

Maximilian Emans[1,2]

[1] Johann Radon Institute for Computational and Applied Mathematics (RICAM)
[2] Industrial Mathematics Competence Center GmbH (IMCC),
4040 Linz, Austria
maximilian.emans@ricam.oeaw.ac.at

Abstract. We present a parallel implementation of an agglomeration scheme for the coarse-grid treatment in algebraic multigrid algorithms for coupled systems. The association of the components of the solution vector with different physical unknowns – bearing particular difficulties to parallel agglomeration techniques – is considered through an appropriate re-ordering of the components of the solution vector. A benchmark of a system of mixed elliptic-hyperbolic character shows that the proposed scheme allows to apply an agglomeration technique which is significantly faster than conventional approaches based on a parallel direct solution of the coarse-grid system.

1 Introduction

Parallel applications of AMG to linear systems reflecting the discretisation of partial differential equations have been demonstrated to be efficient in many areas of science and engineering. Most of the literature refers to scalar systems, i.e. to systems where all components of the solution vector are associated with a single physical unknown. Good scalability of algebraic multigrid (AMG) make these methods also preferred techniques for coupled systems, i.e. for systems where the components of the solution vector are associated with different physical unknowns, e.g. pressure and velocity in fluid flow problems. For this, certain adjustments are necessary, but successful applications are reported e.g. for systems in oil reservoir simulations, see Clees and Ganzer [1]. An important aspect of parallel AMG is the treatment of the coarse grids; it is common practice to employ an iterative or direct solver for a sufficiently small system, see Chow [2]. But in particular if the number of nodes per process is not very large, this practice is often the reason for degraded parallel performance of AMG algorithms. An alternative is an agglomeration scheme where the systems of parallel processes are merged into one which is then solved by one process. It has been shown that this approach is feasible and efficient for scalar systems, see e.g. Emans [3]. For coupled systems, however, an efficient parallel implementation of an agglomeration scheme is not a trivial extension since it should not ignore the association of the components of the unknown vector with the physical unknowns. In this contribution we will present such an efficient implementation and evaluate to what extend and under which circumstances this method is recommendable.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 361–370, 2012.

2 Algorithm and Implementation

Suppose we want to solve the linear problem

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is regular, $\boldsymbol{b} \in \mathbb{R}^n$ is some right-hand side vector and $\boldsymbol{x} \in \mathbb{R}^n$ the solution where n is the rank of A.

The pseudocode of a standard v-cycle AMG algorithm is shown in algorithm 1. The algorithm refers to the solution phase which has to be preceded by a setup phase where the operators $A_l \in \mathbb{R}^{n_l\times n_l}$ $(l = 1, ..., l_{max})$ with system size n_l where $n_{l+1} < n_l$ holds for $l = 1, ..., l_{max} - 1$ are determined. Moreover, a smoother S_l as well as a prolongation operator $P_l \in \mathbb{R}^{n_l\times n_{l+1}}$ and a restriction operator $R_l \in \mathbb{R}^{n_{l+1}\times n_l}$ have to be determined for each level l. It is common practice in algebraic multigrid to choose $R_l = P_l^T$ and to follow the Galerkin approach to generate the coarse-grid hierarchy recursively by

$$A_{l+1} = P_l^T A_l P_l \qquad (l = 1, ..., l_{max} - 1). \tag{2}$$

2.1 Parallel Coarse-Grid Treatment

Algorithm 1 requires the parallel solution of the coarse-grid system. This system is still sparse in the sense that there is a number of zero elements in most of the rows, but the matrix is significantly denser than the fine-grid matrix, see Chow et al. [2]. The size of the system depends on the coarse-grid treatment scheme.

A common approach to the parallel solution of the coarse-grid system is the truncated cycle where the (parallel) coarsening is terminated such that the coarsest grid is distributed to all processes and each process owns at least a few unknowns. This distributed system can then be solved iteratively or directly in parallel. As an iterative solver we use a block-Jacobi scheme where we compute

Algorithm 1. v-cycle AMG

$\boldsymbol{x}_l^{(3)} = \text{AMG\_par}(l, \boldsymbol{r}_l, \boldsymbol{x}_l^{(0)})$
Input: level l, right-hand side $\boldsymbol{r}_l$, initial guess $\boldsymbol{x}_l^{(0)}$
Output: approximate solution $\boldsymbol{x}_l^{(3)}$

1: **if** $l = l_{max}$ **then**
2: (parallel) solution of coarse-grid system: $\boldsymbol{x}_l = A_l^{-1}\boldsymbol{r}_l$
3: **else**
4: (parallel) pre-smoothing: $\boldsymbol{x}_l^{(1)} = S_l(\boldsymbol{r}_l, A_l, \boldsymbol{x}_l^{(0)})$
5: restriction: $\boldsymbol{r}_{l+1} = P_l^T(\boldsymbol{r}_l - A_l\boldsymbol{x}_l^{(1)})$
6: recursive solution of coarse-grid system: $\boldsymbol{x}_{l+1} = \text{AMG}(l + 1, \boldsymbol{r}_{l+1}, \boldsymbol{0})$
7: prolongation of coarse-grid solution and update: $\boldsymbol{x}_l^{(2)} = \boldsymbol{x}_l^{(1)} + P_l\boldsymbol{x}_{l+1}$
8: (parallel) post-smoothing: $\boldsymbol{x}_l^{(3)} = S_l(\boldsymbol{r}_l, A_l, \boldsymbol{x}_l^{(2)})$
9: **end if**

in the setup an LU factorisation of the part of the system matrix that reflects the influences of the unknowns assigned to one process on each other; this can be done entirely in parallel. In the solution phase, the values at the boundaries are exchanged in each iteration and the factorisation is used to compute the solution considering the exchanged values of the neighbouring domains. The advantage of this method is that it is very easy to implement since the exchange routines are identical to those used for the smoothers and the LU factorisation is provided by most linear algebra libraries. The disadvantage is that the accuracy of the solution is hard to control.

As a parallel direct solver for the coarse-grid system we use the existing library MUMPS by Amestoy et al. [4]; this makes an own implementation obsolete. For efficiency the matrix is factorised once in the setup phase while the factors are reused in each iteration. The advantage of this method is that it is safer than an iterative solver since the obtained solution is exact (up to rounding errors).

Both approaches described so far deviate from the idea of multigrid at that point where the solution of the coarse-grid system by recursive application of multigrid is substituted by an iterative or direct solution procedure; this is done although the coarse-grid systems are still of considerable size. The only way to stick more closely to the idea of multigrid is to merge the systems of domains in an appropriate manner such that the coarsest system (of arbitrary small size) can be solved by one process. Instead of terminating the coarsening process at a certain level, the following modified setup applies: Before a new level is constructed on all grids, it is checked if the number of nodes on each grid is larger than the threshold parameter λ. If one grid is smaller, then a merging step is done: Those grids with less nodes than the threshold are merged to the grid of the neighbour with the smallest number of own nodes. This merging step essentially comprises the merging of the system matrices. Its implementation is complex since it is not only necessary to render external influences from the merging neighbours internal ones, but also to update the boundary information of those processes that are not involved in the merger itself, but are adjacent to the merging processes. The algorithm of the solution phase, algorithm 1, has then to be modified, too: Those processes that have no coarse-grid on level $l_{max} + 1$ ("vanishing processes") send their $\boldsymbol{r}_{l+1}$ to an appropriate neighbour and skip step 6 of algorithm 1. The other processes (if they have vanishing neighbours) receive these messages before they perform the coarse-grid correction on level $l + 1$, i.e. before step 6 of algorithm 1, and send the part of the coarse-grid solution $\boldsymbol{x}_{l+1}$ that corresponds to the grid of the vanishing neighbours to these neighbours after step 6 of algorithm 1. Before step 7 the vanishing processes receive these messages as their $\boldsymbol{x}_{l+1}$ and continue.

2.2 AMG for Coupled Systems

For coupled systems with k physical unknowns the system matrix can be split into $k \times k$ blocks where the block A_{ij} reflects the influence of the unknown with index j in the equation associated with the unknown with index i. We denote the number of components of the unknown vector $\boldsymbol{x}$ in eqn. (1) associated with

the physical unknown k by m_k; we assume that the components are ordered by physical unknowns, i.e. the first m_k components are associated with the first physical unknown, the components $m_k + 1, ..., m_k + m_{k+1}$ with the second and so on; for the components of the unknown vector on the initial grid ($l = 1$) this ordering appears to be natural, and for the coarser grids it is natural to keep this order as long as no mergers take place.

$$A = \begin{pmatrix} A_{11} & \dots & A_{1k} \\ \vdots & \ddots & \vdots \\ A_{k1} & \dots & A_{kk} \end{pmatrix} \quad (3) \qquad P = \begin{pmatrix} P_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & P_k \end{pmatrix} \quad (4)$$

The prolongation operator has a similar block-structure. Its non-zero block-pattern reflects the fact that interpolation is only done between nodes associated with the same physical unknown. This is a crucial property of an efficient scheme, see Clees and Ganzer [1].

The application of one of the schemes with a truncated cycle can be implemented easily since for both, the direct solver and the iterative solver the ordering is irrelevant. The straight forward implementation of an agglomeration scheme, however, would merge two systems in such a way that the nodes of the first system keep their indices while the indices of the second system are increased by the number of nodes of the first system, i.e. the unknown vector contains first the unknowns of one system and then those of the second system. This means that they are not ordered by physical unknown but by process.

For the agglomeration scheme there are therefore two principal types of implementations: First, this order by process is maintained and a marker array is defined which carries the information on the physical unknown the node is associated with; this enables a distinction by physical unknown which is necessary to maintain the properties of the prolongation operator. This marker array would also be used in every situation where a distinction by physical unknown is required, e.g. for the computation of the residual norms of the equations associated with the physical unknowns and for the smoothing sweep. This leads to a more complex code where many loops cannot be vectorised since additional if-branches have to be introduced.

The alternative we propose is a reordering of the variables after the merger as shown in the lower part of Fig. 1. Again, it is not trivial to implement since not only the relations within each process need to be adjusted, but the changes also need to be communicated to the neighbours. Moreover, additional book-keeping is required to ensure the correct placement of data during the solution phase if two merging neighbours exchange the right-hand side or the solution vector in the solution phase. The advantage of this concept compared to the approach with the marker field is a more efficient and less complex code of the solution phase.

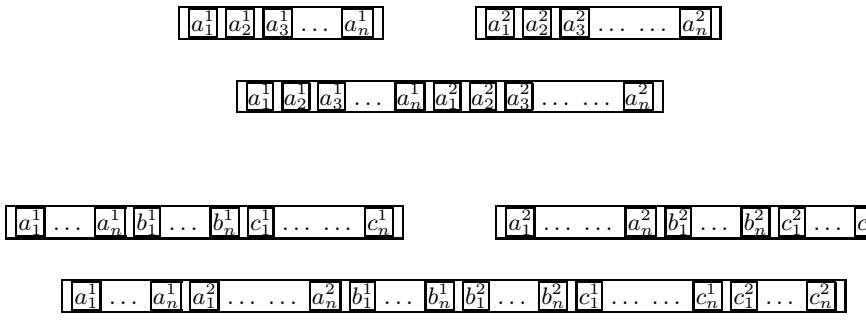


Fig. 1. Order of components of unknown vector before and after merger; top: merger of scalar system (superscript: process, subscript: local numbering before merger), bottom: merger of coupled system (a, b, c: different physical unknowns)

2.3 Implementation Issues

For the the truncated cycle schemes the coarsening process is terminated if one local grid has less than $\lambda = 250$ nodes; the coarsening of the agglomeration scheme is also terminated if the coarsest (global) grid has less than 250 nodes. Grids of less than $\lambda = 250$ nodes are merged. For the iterative coarse-grid solver we use 2 iterations for the block-Jacobi solver for $p \leq 4$, 3 iterations for $p = 8$ and 4 iterations for larger number of processes. The association of the unknowns of the linear system with different physical unknowns is taken into account by the point-based coarse-grid selection scheme, see Clees and Ganzer [1]. The coarse-grid correction schemes are applicable to various AMG schemes with e.g. different coarse-grid selection schemes, cycling strategies, or smoothers. For various AMG schemes the comparison of the coarse-grid techniques leads to similar conclusions; with regard to limited space we have selected a Krylov-accelerated AMG by Notay [5] to draw these conclusions and we skip a comparison to other solvers for similar problems such as they are used e.g. by Starikovičius et al. [6] and refer for such a comparison to Emans et al. [7].

In the Krylov-accelerated AMG, see Notay [5], AMG is used as a preconditioner of an outer Krylov method. The main difference to conventional (fixed cycle) AMG preconditioners such as algorithm 1 is that the coarse-grid system is approximated by one or two iterations (depending on a termination criterion) of an inner Krylov-solver that is preconditioned by AMG instead of being approximated by the multigrid scheme alone. We use the efficient implementation of GCR that Notay [5] suggests as inner and outer Krylov-solver. For this algorithm the double-pairwise aggregation, see again Notay [5], is applied: The aggregates usually comprise four nodes; constant interpolation is applied such that the computation of the coarse-grid system is very cheap since the computation of the elements of A_{l+1} in equation (2) reduces essentially to an addition of rows of A_l.

3 Benchmark

3.1 Background and Details of the Benchmark Case

Our benchmark case is a short period of an unsteady three-dimensional simulation of a full cycle of a four cylinder gasoline engine where the computational domain comprises only one cylinder. The simulated period is part of the compression phase of the engine cycle. The stroke of the cylinder is 81.4 mm, the bore is 79.0 mm yielding a (maximum) volume of 0.4 l (per cylinder). The benchmark case consists of 20 time steps. The computational mesh with 290000 mostly hexagonal cells is shown in Fig. 2. The fuel is octane; the combustion model is based on an Eddy Break-up approach.

The fluid dynamics is modelled by the Navier-Stokes equations amended by a standard k-ε turbulence model; the thermal and thermodynamic effects are considered through the solution of the energy equation for enthalpy and the computation of the material properties using the coefficients of air. The conservation laws are discretised by means of finite volumes on an unstructured mesh. The variable arrangement is collocated and the algorithm managing the coupling of the variables and the non-linearity of the system of partial differential equations is a pressure-enthalpy coupling based on SIMPLE, see Emans et al. [7]. The coupled system we focus our attention on can be written as

$$\begin{pmatrix} C & S_h \\ S_p & H \end{pmatrix} \begin{pmatrix} \boldsymbol{p}' \\ \boldsymbol{h}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{g} \end{pmatrix} \tag{5}$$

where $\boldsymbol{p}'$ and $\boldsymbol{h}'$ are pressure and enthalpy updates, respectively; C and H represent the discretised operators of pressure-correction and energy equation of the SIMPLE scheme, the right-hand sides of these equations are $\boldsymbol{c}$ and $\boldsymbol{g}$, and

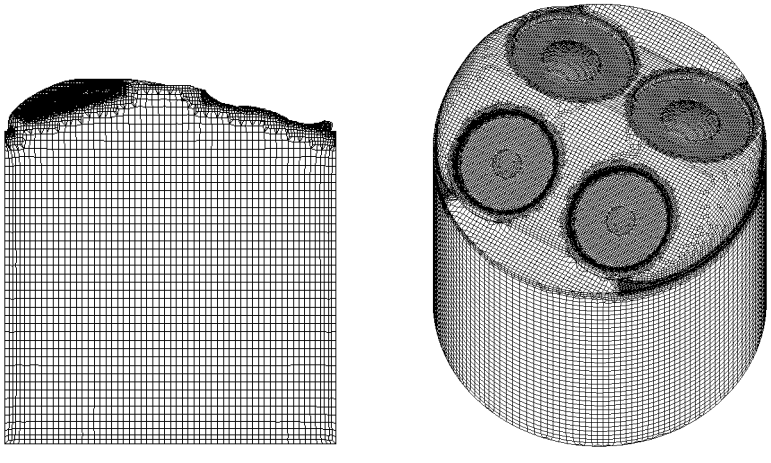


Fig. 2. Slice (left) and surface (right) of the finite volume mesh of the benchmark case

S_h and S_p denote the coupling terms. Since the pressure-correction is an elliptic problem and the energy equation is an hyperbolic one, the underlying system is of mixed elliptic-hyperbolic type. The computing times we report on refer to the solution of this coupled system alone; in each benchmark run 119 different linear systems are treated, i.e. the 1-norm of the residual is reduced by a factor of 1000.

The benchmarks are done within the environment of the software AVL FIRE$^{(R)}$ 2010 which uses the graph partitioning algorithm METIS for the domain decomposition. The linear solver uses the same distribution and receives the linear system in a distributed fashion. The parallelisation of AMG coarsening is simplified: the aggregates are local to each process. The matrices are stored in compressed-row storage (CRS) format, no external libraries (apart from MUMPS) are used. The point-to-point communication relies on an asynchronous communication pattern. For details of the implementation we refer to Emans [3].

For the measurements we used up to four nodes with 2 quad-cores (i.e. 8 cores per node) of a Linux cluster equipped with Intel Xeon CPU X5365 (3.00GHz, main memory 16GB, L2-cache 4MB shared between two cores) connected by Infiniband interconnect (by Mellanox) with an effective bandwidth of around 750 MB/s and a latency of around 3.3 μs. The computational part of the program is compiled by the Intel-FORTRAN compiler 10.1, the communication is performed through calls to platform-MPI subroutines (C-binding). Computations with 1, 2, and 4 processes were done on a single node – unless stated explicitly otherwise –, for 8 and 16 processes we used 2 and 4 nodes respectively. The processes are mapped to the available cores in a way that each process has sole access to one 4MB L2-cache. It is important to note that the MPI implementation uses the shared memory space of one node for the communication between two processes wherever possible. This means that all intra-node communication is done without utilisation of the network interconnect.

3.2 Impact of Coarse-Grid Treatment on Convergence

Since the different coarse-grid treatment methods provide a different correction on the coarsest grids, it is natural that the choice of the method influences the convergence. The left diagram in Fig. 3 contains the accumulated iteration count: it shows that the parallel solution by the direct solver leads to the best convergence. The reason is that this approach guarantees that the system is solved exactly (up to algorithm-related rounding errors) on level l_{max}. The convergence with the iterative coarse-grid solver is the worst among the discussed alternatives; this is a consequence of the lower accuracy of the coarse-grid correction on level l_{max} obtained by the iterative solver. The convergence of the agglomeration approach lies in between the other methods; it does not guarantee the same quality of the solution as the direct solver on level l_{max}, but it transfers an exact solution on the globally coarsest grid by multigrid to this level. The convergence of the agglomeration scheme is almost as good as that of the direct solver scheme.

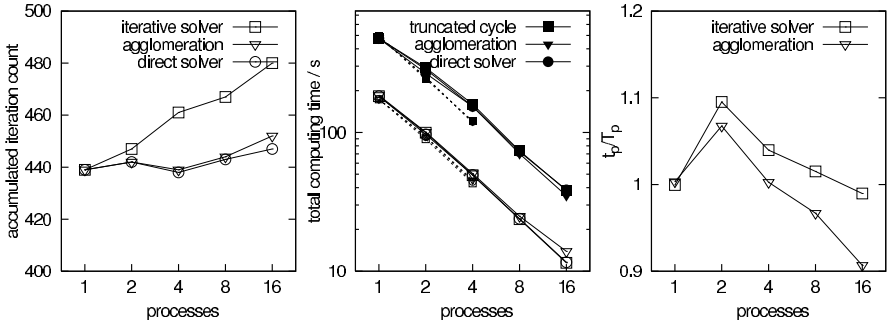


Fig. 3. Accumulated iteration count (left); computing time for setup phase (empty symbols) and solution phase (filled, both middle), small symbols with dashed line: one process per node; ratio t_p/T_p (right)

3.3 Impact of Coarse-Grid Treatment on Performance

In the middle diagram of Fig. 3 the computing times (wall clock times) of the setup phase and total computing time (for the linear solver) are plotted. It can be seen that the setup of the coarse-grid treatment by agglomeration needs more time than that of the alternatives; the difference grows with increasing degree of parallelism. This is due to the fact that the agglomeration requires the construction of additional levels, and due to the merging procedure. Both types of steps include calculation tasks and communication which can be substantial in relation to the cost of typical calculation tasks on these small grids. The computing times of runs with the agglomeration scheme is faster for parallel runs with more than four processes. This is because the solution phase of the agglomeration method is significantly faster than those of the alternatives. We have added the same calculations using one cluster node for each process to the middle diagram, for up to four parallel processes. In the case of four parallel processes this variant is about 30 % faster than our standard process distribution; the difference is significantly larger for the solution phase, but there is hardly a noticeable difference between the coarse-grid treatment schemes.

The right diagram of Fig. 3 shows the ratio of the total computing time t_p with p parallel processes and the total computing time using the parallel direct solver for the coarse-grid treatment T_p with also p parallel processes. We normalise the computing times in this way since a direct solver scheme is used in most AMG implementations, e.g. by Notay [5]. Our benchmark shows that for process numbers of up to four the parallel direct solution of the coarse-grid system is the best choice ($t_p/T_p > 1$). This is the consequence of the fact that this coarse-grid treatment requires the construction of less levels than the agglomeration method (5 compared to 7 in this case) and the direct parallel solver solves a comparatively small system in this case (~ 300 nodes for $p = 2$ compared to ~ 2350 nodes for $p = 16$); equally important is that the fast exchange that occurs through a shared memory mechanism. For a higher degree of parallelism the performance of the

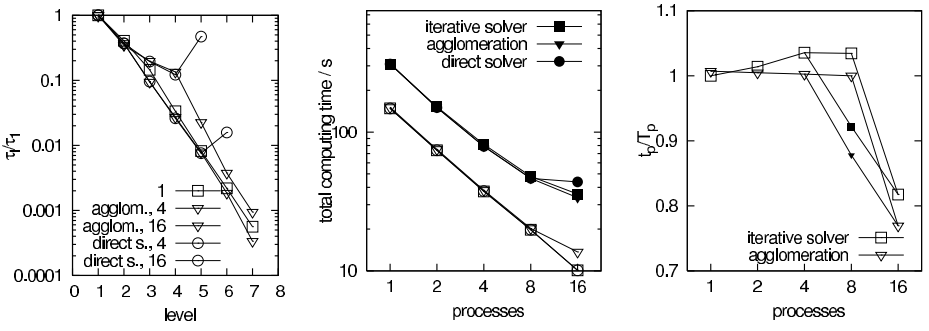


Fig. 4. Ratio τ_l/τ_1 (left; numbers in key: parallel processes), computing time for setup and solution phase (middle; setup: empty symbols, solution: filled symbols; up to eight processes on one machine, 16 processes: 2×8); ratio t_p/T_p (right; empty symbols: 1×1, 1×2, 1×4, 1×8, 2×8; filled symbols: 8 processes distributed 2×4 to two machines)

runs with the parallel direct solver becomes more and more weak compared to that of the other algorithms; this is related to the communication requirements of MUMPS. These results show that the choice of the coarse-grid treatment can reduce (or increase) the total computing time (for setup and solution phase) of parallel AMG algorithms by about 10%.

A very sharp picture of the deviation from the desired typical multigrid behaviour can be obtained if we consider the time τ_l that is spent on all numerical tasks associated with the particular level l (including the transfer of information to and from the next coarser grid, accumulated over all solver calls in a benchmark run). In order to fit the data of runs with different number of processes into one diagram, we have plotted the ratio τ_l/τ_1, see left diagram in Fig. 4; the iterative coarse-grid solver has been skipped since the parallel behaviour of this scheme has already shown to be not satisfying. The curves of the parallel direct solver show a clear increase at the coarsest grid indicating that the time spent on this level does not decrease as it should be the case in ideal multigrid. The agglomeration scheme comes closest to this ideal behaviour which is represented by a straight line in these diagrams. Moreover, the time spent by the direct solver on the coarsest level of this scheme is about one order of magnitude larger than the time spent by the agglomeration scheme on this and all coarser levels!

The identical benchmark on a system of two more recent workstations equipped with two six-core Intel X5670 (3.0 GHz, 6MB L3-cache), connected by an LAN Ethernet with an effective bandwidth of around 11 MB/s and a latency of around 80 μs, reveals the advantage and drawback of the direct solver. Computing times and t_p/T_p for this calculation are shown in Fig. 4. The calculations with eight parallel processes was done twice: Once with all processes on one machine (1×8), and once distributed to two machines as 2×4. If all parallel processes are located on one machine (1×2, 1×4, 1×8) the parallel direct solver is slightly faster than the alternatives. But its performance depends heavily on a fast and

fat connection between the involved processes: therefore the runs with the parallel direct solver are relatively slow as soon as the network interconnect has to be used (2×4 and 2×8). The alternatives suffer far less from the slow network interconnect where the agglomeration scheme is again the fastest method for all AMG variants.

4 Conclusions

The agglomeration scheme, i.e. the solution of the coarsest system on one grid comes closest to the ideal multigrid behaviour. In parallel calculation it shows a much better convergence than truncated cycle schemes with an iterative solver. A parallel direct solution of the distributed coarse-grid problem can be implemented easily using existing libraries. Although the convergence of such schemes appears not to be mitigated by the parallelisation, the parallel performance can be worse than that of the corresponding agglomeration schemes since the direct solution of the coarse-grid system takes up to one order of magnitude longer than the equivalent computational work by the agglomeration scheme. This is true on common cluster hardware with reasonably fast network interconnect and, to a larger extend, on distributed systems with slower network interconnect.

Acknowledgement. Part of this work has been supported in the framework of "Industrielle Kompetenzzentren" by the Austrian Ministerium für Wirtschaft, Jugend und Familie and by the government of Upper Austria.

References

1. Clees, T., Ganzer, L.: An efficient algebraic multigrid solver strategy for adaptive implicit methods in oil reservoir simulation. SPE Journal 15, 670–681 (2007)
2. Chow, E., Falgout, R., Hu, J., Tuminaro, R., Yang, U.: A survey of parallelization techniques for multigrid solvers. In: Heroux, M., Rayhavan, P., Simon, H. (eds.) Parallel Processing for Scientific Computing. SIAM Series on Software, Environments and Tools. SIAM (2006)
3. Emans, M.: Efficient parallel amg methods for approximate solutions of linear systems in CFD applications. SIAM Journal on Scientific Computing 32, 2235–2254 (2010)
4. Amestoy, P.R., Duff, I.S., Koster, J., L'Excellent, J.Y.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications 23, 15–41 (2001)
5. Notay, Y.: An aggregation-based algebraic multigrid method. Electronic Transactions on Numerical Analysis 37, 123–146 (2010)
6. Starikovičius, V., Čiegis, R., Iliev, O.: A parallel solver for optimization of oil filters. Mathematical Modelling and Analysis 16, 326–342 (2011)
7. Emans, M., Žunič, Z., Basara, B., Frolov, S.: A novel SIMPLE-based pressure-enthalpy coupling scheme for engine flow problems. Mathematical Modelling and Analysis 17, 1–20 (2012)

Approaches to Parallelize Pareto Ranking in NSGA-II Algorithm

Algirdas Lančinskas and Julius Žilinskas

Vilnius University, Institute of Mathematics and Informatics,
Akademijos 4, LT-08663 Vilnius, Lithuania
lancinskas@gmail.com, julius.zilinskas@mii.vu.lt

Abstract. In this paper several new approaches to parallelize multi-objective optimization algorithm NSGA-II are proposed, theoretically justified and experimentally evaluated. The proposed strategies are based on the optimization and parallelization of the Pareto ranking part of the algorithm NSGA-II. The speed-up of the proposed strategies have been experimentally investigated and compared with each other as well as with other frequently used strategy on up to 64 processors.

Keywords: Multi-objective Optimization, Pareto Dominance, Pareto Ranking, Non-dominated Sorting Genetic Algorithm.

1 Introduction

There are a lot of fields of industry and science which deal with complex optimization problems. Many of these problems have more than one objective function to optimize, and different functions are often concurrent. Optimization problems of this kind are called *multi-objective optimization problems* (MOPs).

In general a d-dimensional MOP with m objectives

$$f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})$$

is to find a vector

$$\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_d^*)$$

which minimizes (maximizes) the *objective vector*

$$\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})).$$

Here $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is the *decision vector*. Without reducing generality we will deal with MOPs which aim to minimize the objective vector since maximization problem can be easily transformed to minimization problem. In most of the cases it is impossible to minimize all objectives, so there is no single optimal solution to a given MOP, but a set of concurrent solutions exists, known as *Pareto set*. The set of all possible decision vectors satisfying the constraints is known as a *search space* [1]. We denote it by X. The set of all possible objective vectors is called an *objective space*.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 371–380, 2012.

2 Definitions and Notations

2.1 Dominance Relation

Two different decision vectors $\mathbf{a}$ and $\mathbf{b}$ from the search space X can be related to each other in a couple of ways: either one dominates the other or none of them is dominated by the other [2].

Definition. The decision vector $\mathbf{a}$ *dominates* the decision vector $\mathbf{b}$ ($\mathbf{a} \succ \mathbf{b}$) if [2]:

$$f_i(\mathbf{a}) \leq f_i(\mathbf{b}), \quad \forall i = 1, 2, \ldots, m$$

and at least one $j \in (1, 2, \ldots, m)$ exists such that

$$f_j(\mathbf{a}) < f_j(\mathbf{b}).$$

Definition. The decision vector $\mathbf{a}$ *weakly dominates* the decision vector $\mathbf{b}$ ($\mathbf{a} \succeq \mathbf{b}$) if

$$f_i(\mathbf{a}) \leq f_i(\mathbf{b}), \quad \forall i = 1, 2, \ldots, m.$$

Definition. The decision vector $\mathbf{a}$ is called a *dominator* of the decision vector $\mathbf{b}$ if $\mathbf{a} \succ \mathbf{b}$.

Definition. The decision vector $\mathbf{a}$ is said to be *indifferent* to the decision vector $\mathbf{b}$ ($\mathbf{a} \backsim \mathbf{b}$) if

$$\mathbf{a} \not\succeq \mathbf{b} \text{ and } \mathbf{b} \not\succeq \mathbf{a}.$$

The indifference relation is commutative – if $\mathbf{a} \backsim \mathbf{b}$ then $\mathbf{b} \backsim \mathbf{a}$.
The dominance relation has the following properties [3]:

(i) reflexivity: $\mathbf{x} \succeq \mathbf{x}$,
(ii) antisymmetry: $\mathbf{x} \succeq \mathbf{y}$ and $\mathbf{y} \succeq \mathbf{x}$ if and only if $\mathbf{x}$=$\mathbf{y}$,
(iii) transitivity: if $\mathbf{x} \succeq \mathbf{y}$ and $\mathbf{y} \succeq \mathbf{z}$ then $\mathbf{x} \succeq \mathbf{z}$.

2.2 Pareto Ranking

In multi-objective optimization, each decision vector $\mathbf{x}$ is characterized by a vector of objective values $\mathbf{F}(\mathbf{x})$. Many algorithms cannot work with such a vector information and need scalar *fitness values* instead. The fitness values are given by a *fitness function* – a type of objective function that prescribes the optimality of a particular decision vector so that the decision vector may be ranked against all the others. An ideal fitness function correlates closely with the goal of optimization. One of the most used methods to assign fitness values to a particular decision vector is to count the number of its dominators. The function values assigned in this fashion are called *Pareto ranks* and the assignment procedure – *Pareto ranking*. A lower Pareto rank means a better solution fitness and the best possible Pareto rank is 0 which can be assigned to a decision vector from the Pareto set.

Proposition. If the decision vector $\mathbf{x}$ dominates the decision vector $\mathbf{y}$ then the Pareto rank of $\mathbf{x}$ is strictly smaller than the Pareto rank of $\mathbf{y}$.

The proof follows from the transitivity property of the dominance relation.

Let's suppose a set A consisting of N decision vectors. The Pareto ranking procedure of the set A requires to pick the first vector and calculate how many vectors dominate it, then to choose the second one, to do the same and so on. This procedure requires $N(N-1)$ Pareto comparisons.

Let the set A is divided into two sets A_1 and A_2, and $dom(A_1 \otimes A_2)$ denotes the procedure of counting how many dominators each vector from the set A_1 has in the set A_2. Then the set A can be Pareto ranked by carrying out four procedures: $dom(A_1 \otimes A_2)$, $dom(A_2 \otimes A_1)$, $dom(A_1 \otimes A_1)$ and $dom(A_2 \otimes A_2)$. This approach has the same complexity $N(N-1)$.

3 NSGA-II Algorithm

The Non-dominated Sorting Genetic Algorithm (NSGA) was proposed by Srinivas and Deb [4], and was one of the first Multi-Objective Evolutionary (MOE) algorithms which has been applied to various problems [5,6]. An updated version of the algorithm (NSGA-II) was proposed by Deb et al. in [7].

NSGA-II is a genetic algorithm based on obtaining a new child population Q from the parent population P by applying genetic operators (selection, crossover and mutation). Typically both populations P and Q have the same size N. Both populations P and Q are merged into one population R (of the size $2N$), which is Pareto ranked and reduced to the size N by removing vectors with the largest Pareto ranks. In the case of selecting decision vectors with the same rank, a density estimation based on measuring the crowding distance [7] to the surrounding vectors of the same rank is used to choose the most promising solutions [8]. The reduced population R is used as a parent population P for the next generation. The NSGA-II algorithm is given in Algorithm 1.

Algorithm 1

```
begin
   Initialize(P);
   gen = 0;                        // Reset counter of generations
   while (gen < maxgen)
      Q = MakeNewPop(P);           // Create Q using P elements
      R = Q + P;                   // Merge P and Q into one R
      ParetoRanking(R);            // Count dominators of R elements
      P = Reduce(R, N);            // Reduce R and save as P
      gen = gen + 1;               // Increase generations counter
   end;
end.
```

4 Parallelization of the NSGA-II Algorithm

4.1 Parallel Computing

Parallel computing deals with a principle that problems requiring a lot of computational time can be often divided into smaller sub-problems and solved in parallel. The main goal of the parallelism is to reduce execution time of the algorithm utilizing more computing resources.

The performance of a parallel algorithm can be measured by a *speed-up* of the algorithm. The speed-up is denoted by S_p and is defined as a ratio between execution time of the best known sequential algorithm (T_0) and the execution time of the parallel algorithm using p processors (T_p) [9]:

$$S_p = \frac{T_0}{T_p}.$$

The ideal speed-up is $S_p = p$ but it is not always possible to obtain because of reasons such as a time spent for communication between processors or presence of parts of the algorithm which cannot be parallelized (sequential parts). Suppose α is the fraction of the execution time a sequential algorithm spends on non-parallelizable parts. The Amdahl's law [10] states that the maximum speed-up of the parallel algorithm is

$$S_{max} = \frac{1}{\alpha}$$

independent of the number of processors used.

4.2 Strategies to Parallelize NSGA-II

There are some works related to parallelization of NSGA-II algorithm [11,12,13]. Most of them use master-slave strategies and parallelize function evaluations. However the NSGA-II algorithms can be separated into 3 general parts as shown in Fig. 1.

Fig. 1. The general parts of the NSGA-II algorithm

The part (1) – the evaluation of the objective functions – is frequently the largest part of the algorithm and intrinsically parallel. So the easiest way to parallelize the NSGA-II algorithm is to evaluate the objectives by distributing the population among all processors and leave the parts (2) and (3) (see Fig. 1) sequential. We denote this strategy by **Strategy 1**.

This strategy is good if evaluation of objectives requires relatively large amount of time. For example if it requires 99 % of execution time of sequential algorithm the maximum speed-up of the parallel algorithm is 100. But if evaluations of functions require 95 % or 90 % of the execution time the maximum speed-up of the algorithm is respectively 20 or 10. These estimations are based on Amdahl's law (see Section 4.1).

Another part of the algorithm requiring relatively more computational time is part (2) – Pareto ranking and genetic operations. Therefore optimization and parallelization of the part (2) can increase the maximum possible speed-up of the algorithm.

After each generation the new parent population P is created by removing a number of the worst decision vectors. Taking into account the proposition given in Section 2.2 and the fact that removed decision vectors have the largest Pareto ranks, we can state that the number of dominators of the remaining vectors cannot be changed. This imply that we do not need to perform $dom(P \otimes P)$ in any generation except the first. We just need to calculate $dom(P \otimes Q)$, $dom(Q \otimes Q)$ and $dom(Q \otimes P)$. Doing so the number of Pareto comparisons in any generation is reduced by $N(N-1)$ except the first generation which complexity does not changes.

Taking into account this statement, we propose three strategies to parallelize NSGA-II algorithm.

Strategy 2. The population P is sent to all processors following the scheme shown in Fig. 2. Each processor generates child population Q of the size N/p (where p is the number of processors), evaluates objective functions and sends information back to the 1st processor using the corresponding scheme (Fig. 3).

Step(1)	Step(2)	...	Step($\log_2 p$)
$1 \to 2$	$1 \to 3$	...	$1 \to (p/2+1)$
	$2 \to 4$	...	$2 \to (p/2+2)$
		...	...
		...	$(p/2) \to p$

Fig. 2. The scheme for distributing data among the processors

Step(1)	...	Step($\log_2 p$-1)	Step($\log_2 p$)
$(p/2+1) \to 1$	...	$3 \to 1$	$2 \to 1$
$(p/2+2) \to 2$	...	$4 \to 2$	
...	...		
$p \to (p/2)$	...		

Fig. 3. The scheme for gathering data from the processors

In the first step each processor calculates $dom(Q \otimes P)$ and $dom(Q \otimes Q)$. The respective processors send child population Q and the vector of the Pareto ranks of its elements to the respective processors. The receiving processor saves the received population as Q'. In each other step the remaining processors perform $dom(Q \otimes Q')$ and $dom(Q' \otimes Q)$, combine Q and Q' into one double-size Q and send it together with the vector of dominators following the scheme. When the last step is performed the 1st processor additionally has to perform $dom(P \otimes Q)$ in order to finish the Pareto ranking. In the first step processors have to perform

$$\frac{N}{p}\left(\frac{N}{p}-1\right)+\frac{N^2}{p}$$

Pareto comparisons and send

$$\frac{d(N+m)}{p}+\frac{N}{p}$$

double type variables. In the second step processors have to perform

$$2\left(\frac{N}{p}\right)^2$$

Pareto comparison and send

$$2\frac{d(N+m)}{p}+\frac{N}{p}$$

double type variables. In each next step the number of Pareto comparisons increases 4 times and the amount of data to send – 2 times. After the final step the 1st processor has to perform N^2 Pareto comparisons.

In the whole generation processors have to perform

$$\left(\frac{N}{p}\right)^2+\frac{N}{p}+\frac{N^2}{p}+2\left(\frac{N}{p}\right)^2\left(1+2+4+8+\ldots+\frac{p^2}{4}\right)+N^2$$

Pareto comparisons and send

$$\left(d\,(N+m)+\frac{N}{p}\right)(1+2+4+...+\log_2 p)$$

double type variables in $\log_2 p$ data sending procedures, except the first generation, while 1st processor additionally has to perform $dom(P \otimes P)$ which requires $N(N-1)$ Pareto comparison operations.

In this strategy all the processors have to wait while the 1st processor calculates $dom(P \otimes Q)$ at the final step of data transfer. To avoid this we propose Strategy 3, which enables to parallelize the latter operation, but increases the amount of data to send.

Strategy 3. In contrast with Strategy 2, using Strategy 3 in the first step processors additionally calculate $dom(P \otimes Q)$. This require

$$\frac{N^2}{p}$$

Pareto comparisons. Although the processors have additional data (vector of N doubles) which must be transferred to the 1st processor, however it is not necessary to perform N^2 Pareto comparison operations at the end of data transfer procedure.

In both strategies described above the processors generate child subpopulations and send them whole. At the end of generation, a part of child population is rejected. We propose Strategy 4 which rejects some elements of child population before the data sending procedure.

Strategy 4. Performing the procedure $dom(Q \otimes P)$ in the first step of data sending procedure, decision vectors from Q, with Pareto ranks larger than maximum Pareto rank in P, are removed. Doing so there is a probability that the size of Q will be reduced and time required for sending populations and performing Pareto comparisons will be reduced. Note that reducing population 2 times, the number of required Pareto comparisons is decreased by 4 times.

5 Experimental Investigation

The strategies described in the paper have been experimentally investigated by solving benchmark problem ZDT1 [14] with 2 objectives and 10 variables. The size of population was 512 and the number of generations performed – 250. Ten independent runs have been made to estimate the average time of execution. The evaluations of functions last about 90 % of the sequential algorithm execution time. Parallel algorithm has been executed on 2, 4, 8, 16, 32 and 64 processors and the speed-up of the algorithm has been measured. Hardware, containing 16 Intel(R) Core(TM) i5 CPU 760 @ 2.80GHz nodes with 4 cores on each and 4GB of RAM, has been used during experimental investigation.

The results of the experiments (Fig. 4) show that the proposed strategies give significant affect to the speed-up of the algorithm. Using Strategy 1 the speed-up on 64 processors was 8.86 while usage of Strategy 2, Strategy 3 and Strategy 4 increases the speed-up to respectively 12.41, 16.9 and 23.25. Reduction of child population in Strategy 4 gives a significant advantage – the speed-up increases by 37.2 % comparing with the the Strategy 3. The speed-up of the algorithm using Strategy 4 has been calculated with respect to the time of the sequential algorithm with reduction of the child population.

The influence of the time required to evaluate objectives to the speed-up of the algorithm has been investigated by solving MOPs requiring different amount of time to evaluate objectives. It is natural that solving MOP requiring less time to evaluate objectives, the speed-up of the algorithm should be lower. The MOPs, requiring about 80, 70, 60 and 50 percent of sequential algorithm execution time to evaluate the objectives have been chosen for the further investigation. The computations have been performed on 16 processors. The results (Fig. 5) show that reducing the amount of time required to evaluate objectives the speed-up decreases in the same fashion for all strategies. However if only a half of time is utilized to evaluate objectives the proposed strategies still have positive affect to the performance of the parallel algorithm.

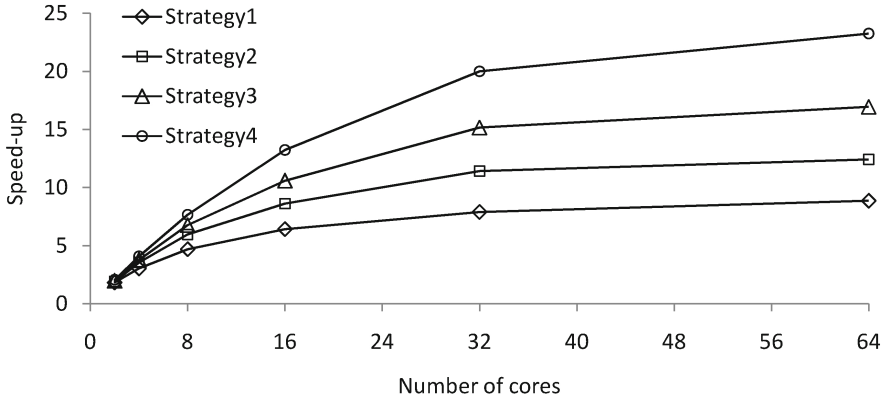


Fig. 4. The speed-up of parallel NSGA-II algorithm using different strategies

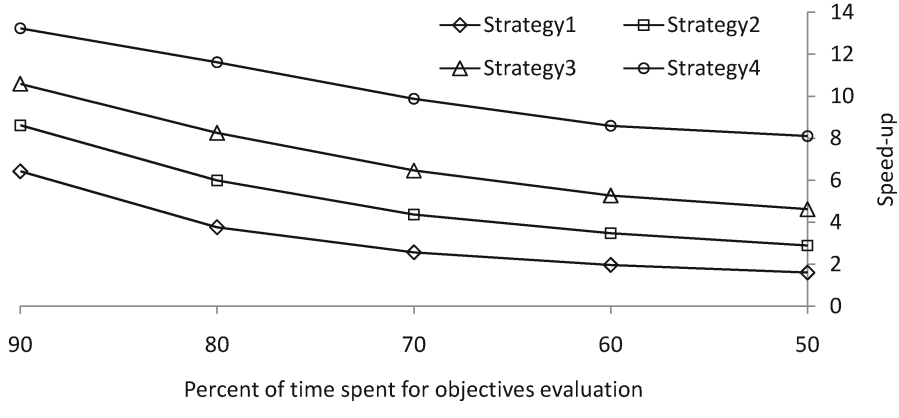


Fig. 5. The influence of the amount of time required to evaluate objectives to the speed-up of the algorithm

The speed-up of the algorithm is

$$S_p = \frac{T_0}{T_p} = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

where α is a fraction of the execution time of non-parallelized part of the sequential algorithm. The values of α have been estimated from the results, and are presented in Table 1. One can see in the table that $\alpha \approx 1 - \beta$ for Strategy 1, where β is the fraction of time spent for objective function evaluations. For the other strategies the values of α are lower what enables increase of the speed-ups.

Table 1. Estimated values of α

	CPU time for objectives				
	90%	80%	70%	60%	50%
Strategy 1	0.096	0.189	0.277	0.368	0.463
Strategy 2	0.055	0.094	0.135	0.179	0.227
Strategy 3	0.032	0.050	0.070	0.095	0.117
Strategy 4	0.014	0.025	0.041	0.058	0.065

6 Conclusions

In this paper we have presented the approach to optimize the Pareto ranking in the Non-dominated Sorting Genetic Algorithm (NSGA-II) and make the algorithm more suitable to parallelization. Also we have presented four strategies to parallelize NSGA-II algorithm – the frequently used one which parallelize the evaluation of objectives and three proposed which parallelize the Pareto ranking and optimize communication between the processors. All strategies have been experimentally investigated by solving multi-objective optimization problems using up to 64 processors. The results of the experimental investigation show that the proposed strategies give positive affect to the speed-up of the algorithm – the execution time has been reduced 23.25 times. Decreasing the amount of time required to evaluate objectives the speed-up decreases in the same fashion for all strategies used but the proposed strategies have positive affect to the performance of the algorithm independent of the amount of time required to evaluate objectives.

Acknowledgement. This work was funded by the Research Council of Lithuania (project number MIP-108/2010).

References

1. Durillo, J.J., Nebro, A.J., Coello, C.A.C., García-Nieto, J., Luna, F., Alba, E.: A Study of Multiobjective Metaheuristics When Solving Parameter Scalable Problems. IEEE Transactions on Evolutionary Computation 14(4), 618–635 (1981)
2. Sbalzarini, I.F., Muler, S., Koumoutsakos, P.: Multiobjective optimization using Evolutionary Algorithms. In: Proceedings of the Summer Program 2000, Center of Turbulence Research, pp. 63–74 (2000)
3. Voorneveld, M.: Characterization of Pareto Dominance. SSE/EFI Working Paper Series in Economics and Finance, No. 487 (2002)
4. Srinivas, N., Deb, K.: Multi-Objective Function Optimization Using Non-dominated Sorting Genetic Algorithms. Evolutionary Computation 2(3), 221–248 (1995)
5. Mitra, K., Deb, K., Gupta, S.K.: Multiobjective Dynamic Optimization of an Industrial Nylon 6 Semibatch Reactor Using Genetic Algorithms. Journal of Applied Polymer Science 69(1), 69–87 (1998)

6. Weile, D.S., Michielssen, E., Goldberg, D.E.: Genetic Algorithm Design of Pareto-optimal Broad Band Microwave Absorbers. IEEE Transactions on Electromagnetic Compatibility 38(3), 518–525 (1996)
7. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 849–858. Springer, Heidelberg (2000)
8. Durillo, J.J., Nebro, A.J., Luna, F., Alba, E.: A study of master-slave approaches to parallelize NSGA-II. In: IEEE International Symposium on Parallel and Distributed Processing, pp. 14–18 (2008)
9. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing, 2nd edn. Pearson Education Limited, Edinburgh (2003)
10. Amdahl, G.M.: Validity of the single-processor approach to achieving large scale computing capabilities. In: AFIPS Conference Proceedings, vol. 30, pp. 483–485 (1967)
11. Branke, J., Schmeck, H., Deb, K., Reddy, M.: Parallelizing Multi-Objective Evolutionary Algorithms: Cone Separation. In: 2004 Congress on Evolutionary Computation (CEC 2004), pp. 1952–1957 (2004)
12. Deb, K., Zope, P., Jain, S.: Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 534–549. Springer, Heidelberg (2003)
13. Streichert, F., Ulmer, H., Zell, A.: Parallelization of Multi-objective Evolutionary Algorithms Using Clustering Algorithms. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 92–107. Springer, Heidelberg (2005)
14. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation 8(2), 173–195 (2000)

OpenCL Implementation of Cellular Automata Finite Element (CAFE) Method

Lukasz Rauch, Krzysztof Bzowski, and Artur Rodzaj

AGH - University of Science and Technology,
al. Mickiewicza 30, 30-059 Kraków, Poland
{lrauch,kbzowski}@agh.edu.pl

Abstract. The implementation of multiscale numerical simulations on heterogeneous hardware architectures is presented in the paper. The simulations are composed of coupled micro and macro scale approaches, which are implemented by using cellular automata and finite element method respectively. Details of both of these methods are described in the papers as well. The simulations were performed for the problem of material heat treatment (macro scale) with simultaneous application of grain growth calculation (micro scale). Comparison of quantitative results obtained by using separated and coupled computing methods are presented in form of speedup and efficiency coefficients.

Keywords: heterogeneous computing, multiscale modelling, finite element method, cellular automata.

1 Introduction

The development of reliable material models for metal forming simulations has been in interest of scientists for a number of years. A series of deterministic models, based either on closed form equations describing the flow stress dependence on external variables [1], or on differential equations describing the evolution of internal variables [2] have been published in the scientific literature. These models are commonly used in simulations for the majority of metal forming processes and give reasonably accurate results in macro scale. However, they fail to describe the material behaviour when some stochastic material phenomena occur in micro scale. These phenomena are closely coupled and depend on each other giving strong impact to higher scales. Therefore, multiscale modelling methods are introduced to take into account material behaviour in many scales simultaneously. The multiscale modelling of material behaviour, during processes of heating and cooling or under loading conditions, is a highly sophisticated numerical problem. It requires many advanced algorithms solving a thermo-mechanical issues in macro scale as well as algorithms simulating undergoing processes of microstructure evolution e.g. strain localization [3], recrystallization [4] or phase transformation [5] occurring in micro and nano scales. In many cases the coupled Cellular Automata (CA) and Finite Element (FE) methods, as one complex CAFE approach, are applied for numerical simulations [6], where the CA

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 381–390, 2012.

and FE methods are used in micro and macro scales respectively. However, the mentioned methods are very time and memory consuming, which is caused by detailed material representation and iterative character of calculations in both scales. Many improvements were proposed to parallelize this computational process [7,8]. A reliable review on efficient parallelization of FE is also presented in [9]. Nevertheless, the most of the approaches, published in recent literature, solve the computational problems in each scale separately and none of these methods analyses the multiscale computing as one complex approach. On the other hand, the recent trend on green computing [10], which aims at maximizing computing efficiency with simultaneous energy saving, forces to create new advanced parallel solutions, which are based on heterogeneous hardware architectures like CPU-GPU, Cell Broadband Engine Architecture or Field Programmable Gate Arrays [11]. Thus, great majority of recent implementations of various numerical algorithms is based on OpenCL standard [12]. This paper presents the details of CAFE approach implemented using OpenCL standard as well. The following section contains CA module description including architecture, kernel generation and obtained results. The third section presents finite element method implementation, and participation of host and OpenCL device in calculations. The combination of both of these methods and obtained quantitative results from multiscale computing are presented in fourth section.

2 Realization of CA Module

CA module is responsible for numerical simulations of material phenomena in microscale. The architecture of this part of the software was prepared to be used as standalone computer system as well as an element of multiscale simulations coupled with FEM module. Moreover, the usage of this module as programming framework [13] was a very important aspect, which influenced the final structure of classes and their functionality.

2.1 Functional Assumptions

CA are a powerful tool for solving sophisticated mathematical problems by defining simple variables and rules applied on a space of neighbouring elements. Moreover, algorithms based on CA are usually very well parallelized as Single Program Multiple Data (SPMD) according to Flynn's taxonomy. This makes them acceptable for processing units composed of many computational cores with shared access to global memory. Similar situation takes place in the case of modelling of material phenomena, where a CA space represents discretized material sample and particular models are described as a set of rules combined in form of one program. Such program is performed as a single thread representing behaviour of one cell, which uses its own variables and data possessed by its neighbours. The package of these data has to be synchronized with other threads. That is why the logic and structure of the CA module are so crucial in planning of device resources usage. The second reason is the bottleneck related

to data transfer between host and OpenCL device, which is associated with a significant time delay, influencing the efficiency of analysed approach. However, in this paper the data transfers are not taken into account during assessment of proposed approach efficiency.

2.2 Implementation Details

The architecture of CA classes is divided according to their functionality into classes dedicated to constructing variables and rules of automaton, supporting OpenCL kernels generation and managing the usage of devices memory. This approach allows material engineers to define the algorithms modelling material behaviour in micro scale and relieves them of thinking about efficiency, parallelization and hardware architecture. This functionality is covered by MemoryManager class, which responses for organizing device resources and rewriting of memory space for data arrays. These arrays originate from one of the OpenCL constraints allowing the upstreaming of data to the device exclusively in form of primitive types or arrays of primitive types. This restriction forces to rewrite the object-oriented model into arrays of integers or floats. The classes within the framework allow also to avoid the redundancy of data on different levels of material description. Similar solution is presented in other Authors' work [13], where CA framework for multicore CPU processors is proposed). The assumed OpenCL standard forces kernels to be implemented in specific language based on C99. The DeviceRule class, which extends the Rule class is responsible for convertion of C++ into C99 implementation, providing a bridge between a model and a final code for heterogeneous hardware architectures. KernelManager class combines all converted codes into a kernel, compile it and finally deploy it on target OpenCL device.

2.3 Model Implementation and Kernel Creation

For the purposes of this paper the phenomenon of simple metallic material grains growth in high temperatures was selected. It assumes that during heating of material in high temperatures, particular grains inside material microstructure change their volume dependently on current temperature and a curvature of grain boundary. This phenomenon is reflected in CA rules by introducing the probability of boundary motion for each cell using the following equation:

$$p = e^{\frac{Q_b}{TR}} \frac{k}{k_{max}} \tag{1}$$

where R - the universal gas constant, T - a current temperature, Q_b - a boundary diffusion activation energy, k_{max} - number of cells belonging to different grain, k - number of neighbours. This material behaviour is very well known by scientists, thus, it can be easily implemented and verified, offering reliable qualitative and quantitative results [14]. The source codes listed and described in this section present examples of implementation of kernel generation for this particular model.

Listing 1. The generated kernel of grain growth

```
  #define NUM_CELL_REALS 1
  #define NUM_GRAIN_INTS 5
  #define DIMENSION 2
  #define X_SIZE 1000
  #define NUMBER_OF_CELLS 1000000
// random generator [15]
  rand = MWC64X_NextUint(&rng);
//temperature
  float Tem = 273.15 + T[nr];
// grain boundary mobility
  float m = exp(-1 * (float)QB / (R * Tem));
// grain growth condition
  if((float)( 20.0 * m * ((float)ngr_s / 9.0)) > rand)
      GRAIN_W(gx, gy) = nr + 1;
  else
      GRAIN_W(gx, gy) = GRAIN(gx, gy);
```

The creation of the model begins from a definition of CA space parameters, number of grains and initial values of internal variables. The process of the initialization is held in the InitEvent, which is followed by specification of the neighbourhood scheme. Then, the constructor of DeviceRule class is activated to convert the C++ and pseudo codes model into OpenCL kernel (listing 1). This kernel is the most essential part of computing process.

2.4 Performance

The specific character of the CA algorithms, including large number of conditional statements as well as probabilistic dependencies between cells, makes it almost impossible to reliably estimate performance in Gflops. Therefore, speedup and efficiency were estimated on the basis of time measurements, which do not include transfer of data from host to device memory. However, times of internal data transfer from global to local memory of the device are included in this analysis, which for two dimensional memory allocation (in case of CA space) have huge impact on final results (table1).

The calculations were performed on NVIDIA GTS 250 with 16 multiprocessors, 8 streaming processors (SP) each. The presented differences in time results for the same size of local memory, but different x and y dimensions, is probably caused by specific data alignment in global device memory. Nevertheless, this conclusion is not confirmed directly. The obtained speedup and efficiency are presented in figure 1.

Table 1. Time of calculations for Grain Growth algorithm for 1000x1000 CA space

Local Size	**Time** (s)	**Best time** (s)
1x1	212.20	7212.20
1x2, 2x1	147.09, 194.93	147.09
1x4, 4x1, 2x2	180.74, 96.06, 138.07	96.09
1x8, 8x1, 2x4, 4x2	114.26, 58.66, 92.93, 131.97	58.66
2x8, 8x2, 4x4	57.91, 79.35, 87.84	57.91
4x8, 8x4	56.54, 53.16	53.16
8x8	33.79	33.79

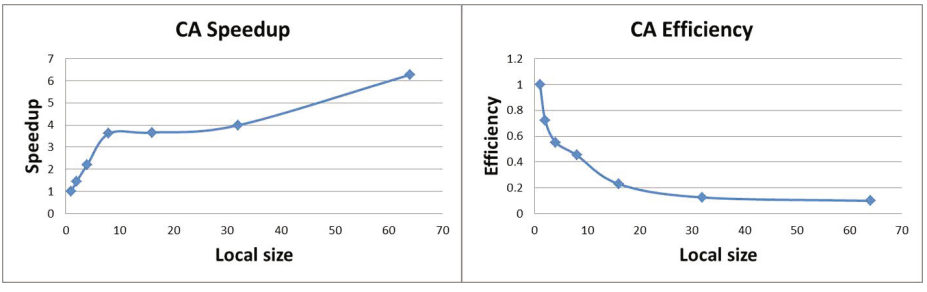


Fig. 1. Estimated speedup and efficiency of OpenCL based CA module

3 Realization of FE Module

3.1 Problem Definition

Heat transfer problem was selected as an example of macro scale problem, which influences other scales. It is solved by using the Finite Element Method. Distributed characteristics of temperature fields in transient form are obtained by solving linear systems of partial differential equations:

$$[H]\{T\} + [C]\frac{\partial}{\partial t}\{T\} + \{P\} = 0 \tag{2}$$

where $[H]$ - heat conductivity matrix, $[C]$ - heat capacity matrix, $\{P\}$ - heat load vector, $\{T\}$ - nodal temperatures vector, t - time. The analysis of temperature field was performed with 8-nodes square elements and integration scheme based on 9-point Gaussian quadrature. Heat load vector, matrix capacity and thermal conductivity matrices are defined by the following equations:

$$[H] = \int_v k\left(\left(\frac{\partial\{N\}}{\partial x}\right)\left(\frac{\partial\{N\}}{\partial x}\right)^T + \left(\frac{\partial\{N\}}{\partial y}\right)\left(\frac{\partial\{N\}}{\partial y}\right)^T\right)dV + \int_s \alpha\{N\}\{N\}^T dS \tag{3}$$

where k is a conductivity coefficient, N - matrix of shape functions, α - convection coefficient.

$$[C] = \int_v c\rho\{N\}\{N\}^T dV \tag{4}$$

where c is a specific heat, ρ - material density.

$$\{P\} = \int_s (-\alpha T_\infty + q)\{N\} dS \tag{5}$$

where T_∞ in an ambient temperature and q is a heat flux. Values of each element of local matrices are placed in the appropriate places of global system of equations. This procedure is performed in accordance to the number of nodes in the corresponding element. Obtained sparse and symmetric systems of linear equations can be efficiently solved using conjugate gradient method.

3.2 Implementation Details

The main workflow of FE module is presented in figure 2. All operations related to FE mesh generation, pre-processing activities and assembly of the process are performed on host side.

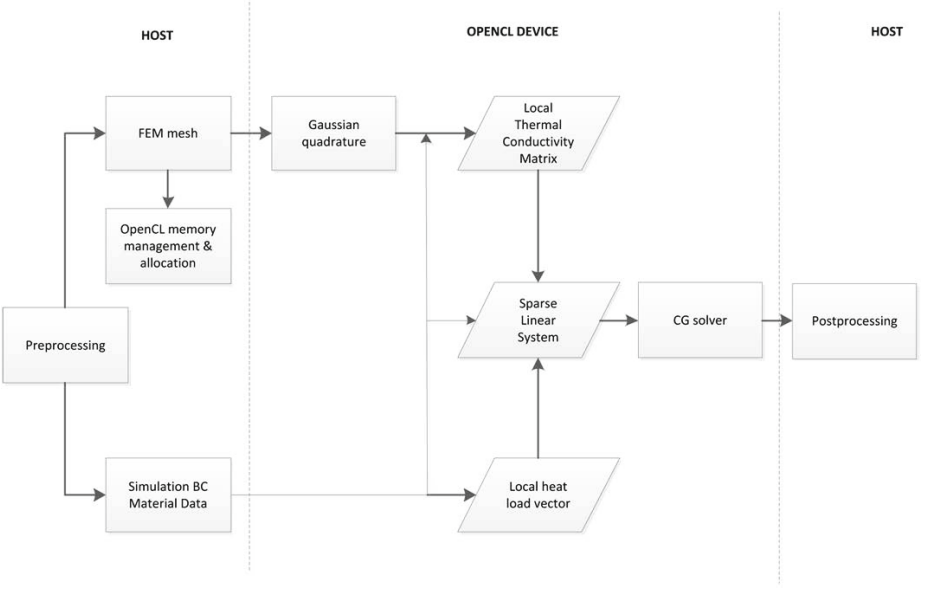


Fig. 2. The main workflow of FE module

The model does not implement grid deformation, which avoids necessity of mesh refinement during calculations. This assumption allows single allocation of an OpenCL device memory for sparse linear system. The Compressed Row Storage (CRS) algorithm is applied to reduce the space held by the data and,

simultaneously, to optimize a time required for data transfer between a host and a device, being the key issue for gaining best performance. Gaussian quadrature used to obtain components of the element's matrices as well as the assembly of global systems are performed on OpenCL device. Heterogeneous nature of the model was obtained by controlling the flow of data by the host, and leaving all of the numerical operations for OpenCL device. The necessary mechanisms associated with memory management, matrix compression and the solution of equations are provided by Vienna Computing Library (ViennaCL) engine, which is a scientific computing library written in C++ and based on OpenCL standard, allowing high-level access to the vast computing resources available on parallel hardware architectures. Moreover, it supports common linear algebra operations (BLAS levels 1, 2 and 3) and solving of large systems of equations by means of iterative methods with optional preconditioner.

3.3 Performance

The tests were performed using NVIDIA GTX 260 for different values of global size parameter, determining number of cores active during the calculations. Obtained results shown satisfactory parallelization (figure 3) for this type of sophisticated numerical problems. Nevertheless, the overall performance not exceeding 2GFlops requires further improvements and optimization of source codes.

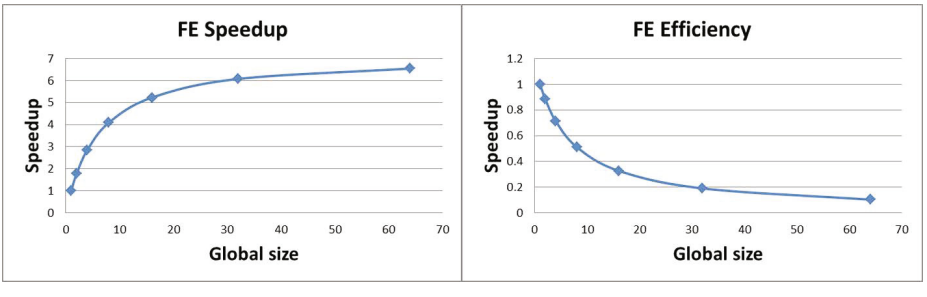


Fig. 3. Estimated speedup and efficiency of OpenCL based FE module

4 Coupled CAFE Modelling

The CA and FE modules presented in previous sections were coupled together to implement complex multiscale approach. The main workflow (figure 4) assumes separated initialization of CA and FE. This process is followed by the performance of macro scale calculations, which is justified by the behavior of a real material, where microstructural phenomena are induced by macro scale temperatures.

The main iteration of CAFE consists of subsequent performance of FE and CA calculations, where FE influences CA by passing temperature values and CA

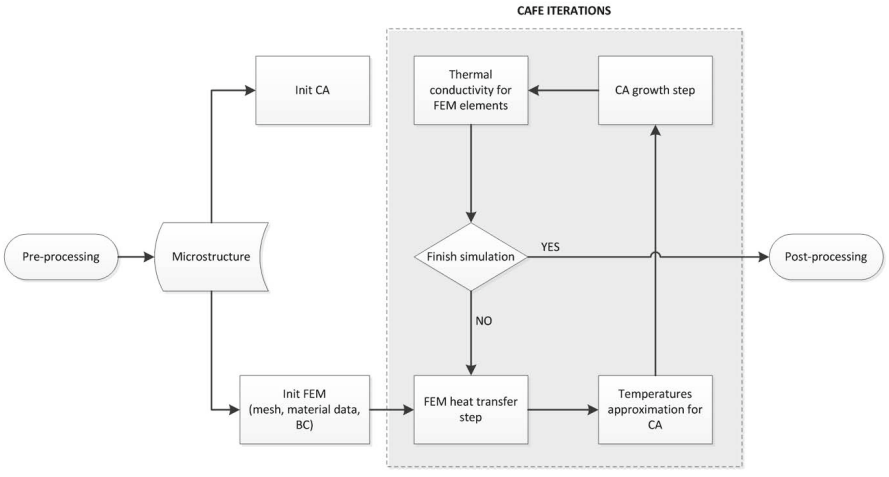


Fig. 4. The main workflow of coupled CAFE approach

influences FE by returning modified coefficients of material thermal conductivity. All computational issues are performed on OpenCL device, while a host is responsible for post-processing, visualization and analysis of results.

5 Results

The obtained qualitative results from coupled CAFE model were verified and high reliability of implemented method was achieved. Therefore, in this section only quantitative results (figure 5) are discussed. Two different graphic card i.e. GTS250 and GTX260 were used for CA and FE separately to check the influence of the device architecture on final speedup and efficiency. It occurred that CA obtained better results on GTS250 and FE on GTX260. This is the initial step to further investigation on determination of the best architecture for given problem. The CAFE approach was tested on both devices as well, however it obtained better efficiency on GTX260 (figure 5).

The measured timings shown that, besides computing procedures, the data transfers between local and global device memories take a part in final efficiency. However, the major influence on CAFE speedup and efficiency is contributed by FE module, which is a result of very good parallelization of the main solver. The applied FE solver achieves maximum speedup at 11.22 (this is only the feature of Conjugate Gradient solver), while the same coefficient in the case of CA model does not exceeded 6.3 with 0.1 efficiency. Therefore, the next step of research will be focused on localization of all bottlenecks in the implemented software by analysis of isoefficiency.

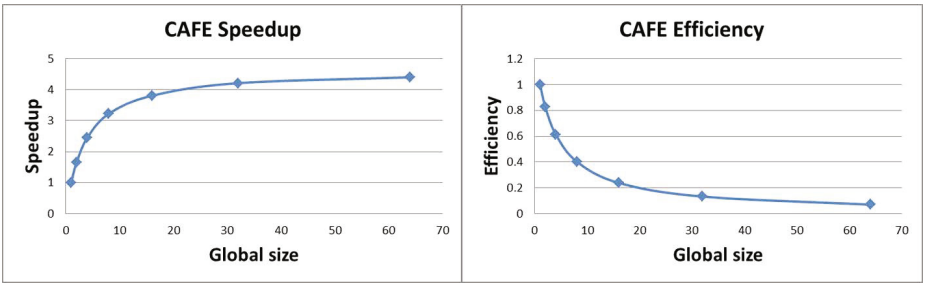


Fig. 5. Estimated speedup and efficiency of OpenCL based CAFE

6 Conclusions

The OpenCL implementation of coupled CA and FE solvers was presented in the paper. The qualitative results of material phenomena modelling proved high reliability in comparison to results of real experiments. The obtained quantitative results (timings, speedups and efficiencies) lead to two main conclusions:

- Coupled CAFE approach can be implemented on heterogeneous hardware architectures, like GPGPUs, offering satisfactory performance, especially in comparison to conventional CPU implementations.
- The definition of x and y local memory sizes in two dimensional problem (CA model) influences times of performance, which may be caused by specific alignment of data in a global memory and accompanying latencies in data transfer.

The next step of the development of proposed approach assumes also addition of nano scale modelling method as well as an attempt to estimation of performance in GFlops.

Acknowledgments. The financial support of the National Centre for Research and Development (NCBiR)project no. R07 0006 10 is acknowledged.

References

1. Grosman, F.: Application of the flow stress function in programmes for computer simulation of plastic working processes. Journal of Materials Processing Technology 64, 169–180 (1997)
2. Estrin, Y., Mecking, H.A.: Unified phenomenological description of work hardening and creep based on one-parameter models. Acta Metallurgica 32, 57–70 (1984)
3. Madej, L., Rauch, L., Yang, C.: Strain distribution analysis based on the digital material representation. Archives of Metallurgy and Materials 54, 499–507 (2009)
4. Goetz, R.L., Seetharaman, V.: Modeling dynamic recrystalization using cellular automata. Scripta Materialia 38, 405–413 (1998)

5. Pietrzyk, M., Madej, L., Rauch, L., Golab, R.: Multiscale modeling of microstructure evolution during laminar cooling of hot rolled DP steels. Archives of Civil and Mechanical Engineering 10, 57–67 (2010)
6. Gawad, J., Pietrzyk, M.: Application of CAFE coupled model to description of microstructure development during dynamic recrystallization. Archives of Metallurgy and Materials 52, 257–266 (2007)
7. Cannataro, M., Di Gregorio, S., Rongo, R., Spataro, W., Spezzano, G., Talia, D.: A parallel cellular automata environment on multicomputers for computational science. Parallel Computing 21, 803–823 (1995)
8. Patra, A.K., Laszloffy, A., Long, J.: Data structures and load balancing for parallel adaptive hp finite-element methods. Computers & Mathematics with Applications 46(1), 105–123 (2003)
9. 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)
10. Murugesan, S.: Harnessing Green IT: Principle and Practices. IT Professional 10(1), 24–33 (2008)
11. Brodtkorb, A.R., Dyken, C., Hagen, T.R., Hjelmervik, J.M., Storaasli, O.O.: State-of-the-art in heterogeneous computing. Scientific Programming 18, 1–33 (2010)
12. Maciol, P., Plaszewski, P., Banas, K.: 3D finite element numerical integration on GPUs. Procedia Computer Science 1, 1093–1100 (2010)
13. Spytkowski, P., Klimek, T., Rauch, L., Madej, L.: Implementation of cellular automata framework dedicated to digital material representation. Computer Methods in Materials Science 9(2), 283–288 (2009)
14. Chen, F., Cui, Z., Liu, J., Chen, W., Chen, S.: Mesoscale simulation of the high-temperature austenizing and dynamic recrystallization by coupling a cellular automaton with a topology deformation technique. Materials Science and Engineering A 527, 5539–5549 (2010)
15. Thomas, D.B., Howes, L., Luk, W.: A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation. In: Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2009, pp. 63–72 (2009)

Parallelization of EULAG Model on Multicore Architectures with GPU Accelerators

Krzysztof Rojek and Lukasz Szustak

Czestochowa University of Technology
Dabrowskiego 69, 42-201 Czestochowa, Poland
{krojek,lszustak}@icis.pcz.pl

Abstract. EULAG (Eulerian/semi-Lagrangian fluid solver) is an established computational model developed by the group headed by Piotr K. Smolarkiewicz for simulating thermo-fluid flows across a wide range of scales and physical scenarios.

In this paper we focus on development of the most time-consuming calculations of the EULAG model, which is multidimensional positive definite advection transport algorithm (MPDATA). Our work consists of two parts. The first part is based on the GPU parallelization using ATI Radeon HD 5870 GPU, NVIDIA Tesla C1060 GPU, and Fermi based NVIDIA Tesla M2070-Q, while the second one assumes the multicore CPU parallelization using AMD Phenom II X6 CPU, and Intel Xeon E3-1200 CPU with Sandy Bridge architecture. In our work, we use such standards for multicore and GPGPU programming as OpenCL and OpenMP.

The GPU parallelization is based on decomposition of the algorithm into several smaller tasks called kernels. They are executed in a FIFO order corresponding to the dependency tree expressing data dependencies between kernels. To optimize performance of the resulting implementation, we utilize the extensive vectorization of each kernel, as well as overlapping of data transfer with computations.

At the same time, when considering CPU parallelization we focus on multicore processing, vectorization and cache reusing. To achieve high efficiency of computations, the SIMD processing is applied using standard SSE and new AVX extensions. In this paper we provide performance analysis based on the Roofline Model, which shows inherent hardware limitations for MPDATA, as well as potential benefit and priority of optimizations. In order to alleviate memory bottleneck and improve efficient cache reusing, we propose to use the loop tiling technique.

1 Introduction

EULAG [3,5] (Eulerian/semi-Lagrangian fluid solver) is an established computational model developed by the group headed by Piotr K. Smolarkiewicz for simulating thermo-fluid flows across a wide range of scales and physical scenarios, such as numerical weather and climate prediction. EULAG is a representative of the class of anelastic hydrodynamic models.

Preliminary studies of porting anelastic numerical models to modern architectures, including GPUs, were carried out in work [10]. Selected parts of this

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 391–400, 2012.

model were ported to ATI Radeon HD 5870 and NVIDIA Tesla C1060 cards. The achieved performance results show the potential gains in computing performance on modern computer architectures. The problem of adapting the EULAG model to modern hardware architectures was also brought up in [4]. The results achieved for porting selected parts of EULAG to NVIDIA GPUs, using an automatic approach as well, unveil potential in running scientific applications, including anelastic numerical models, on novel hardware architectures.

In this paper, we focus on parallelization of the most time-consuming algorithm of the EULAG model, which is MPDATA [10]. Our work consists of two parts. The first part is based on the GPU parallelization using NVIDIA and AMD architectures, while the second one assumes the multicore CPU implementation. In our work, we use such standards for multicore and GPGPU programming as OpenCL and OpenMP.

2 Architecture Overview

Our research is based on two kind of architectures. The first one are GPUs, while the second one are CPUs. We focus only on features of these architecture which are used in our work.

2.1 Architecture of GPUs

Our research is focused on NVIDIA Tesla C1060 and M2070-Q, as well as ATI Radeon HD 5870.

Architecture of NVIDIA Tesla. NVIDIA Tesla C1060 [6] includes 10 Thread Processing Clusters (TPC). Every TPC contains 3 compute units. Each compute unit consists of 8 processing elements, and 16KB of local memory. It gives a total number of 240 available processing elements with a clock rate of 1296 MHz. It provides a peak performance of $240 * 1.296 * 2 = 0.622$ Tflop/s in single precision. This graphics accelerator card includes 4 GB of global memory with the peak bandwidth of 102.4 GB/s.

NVIDIA Tesla M2070-Q [6] is based on Fermi architecture, that supports fully coherent L2 cache. It contains 448 available processing elements with a clock rate of 1147 MHz, so the peak performance is $448 * 1.147 * 2 = 1.03$ Tflop/s in single precision. This graphics accelerator card includes 6 GB of global memory with the peak bandwidth of 148.0 GB/s.

Architecture of ATI Radeon HD 5870. ATI Radeon HD 5870 [2] includes 20 compute units. Each compute unit consists of 16 processing elements, and 32KB of local memory. Each of the processing element is built of 5 streaming processors. It gives a total number of 1600 available streaming processors with a clock rate of 850 MHz, and provides the peak performance of $1600 * 0.850 * 2 = 2.72$ Tflop/s in single precision. This accelerator card includes only 1 GB of global memory with the peak bandwidth of 153.6 GB/s.

2.2 Architecture of CPUs

Our research is based on AMD Phenom II X6 CPU, and Intel Xeon E3-1200 CPU.

Architecture of AMD Phenom II X6. AMD Phenom II X6 CPU [1] contains of six cores with clock frequency of 3.2GHz. This CPU supports Streaming SIMD Extensions (SSE), which can greatly increase performance when exactly the same operations are to be performed on multiple data objects. Therefore, the peak performance of AMD Phenom II X6 processor is $3.2 * 6 * 4 * 2 = 153.6$ Gflop/s in single precision with SSE enabled, and $3.2 * 4 * 2 = 38.4$ Gflop/s without SSE.

Architecture of Intel Xeon E3-1200. Intel Xeon E3-1200 [8] is based on Sandy Bridge architecture, which is the first implementation of Intel Advanced Vector Extensions (AVX). This processor includes four cores with clock frequency of 3.4GHz. For Intel Xeon E3-1200 processor with AVX enabled, the theoretical performance is $3.4 * 4 * 8 * 2 = 217.6$ Gflop/s in single precision, and only $3.4 * 4 * 2 = 27.2$ Gflop/s without AVX.

Intel Advanced Vector Extensions Overview. Before Sandy Bridge Intel microarchitecture, the SIMD vectorization was provided by the Intel Streaming SIMD Extensions (Intel SSE). Intel SSE instructions use eight 128-bit registers where uniform type data can be packed, and enable operating on 4 float elements per iteration instead of a single element. The Intel AVX [8] offers a significant increase in the floating-point performance over previous generations of 128-bit SIMD instruction set extensions. AVX increases the number of registers from 8 to 16 and width of the registers from 128 bits to 256 bits. The new ability to work with 256-bit vectors enables operating on 8 float or 4 double elements per iteration, instead of a single element.

3 The Scope of Our Research on the EULAG Model

One of the most time-consuming calculations [10] of the EULAG model is multidimensional positive definite advection transport algorithm (MPDATA). In this work, we take into account the linear version of MPDATA, which is based on the following equation [7]:

$$\Psi_i^{n+1} = \Psi_i^n - \frac{\delta t}{\nu_i} \sum_{j=1}^{l(i)} F_j^{\perp} S_j, \tag{1}$$

where Ψ is a nondiffusive scalar field, S_j refers both to the face itself and its surface area, ν_i is the volume of the cell containing vertex i, while $F_j^{\perp}$ is interpreted as the mean normal flux through the cell face S_j averaged over temporal increment δt.

The approximation of surface area S begins with specifying fluxes $F_j^{\perp}$:

$$F_j^{\perp} = 0.5(v_j^{\perp} + |v_j^{\perp}|)\Psi_i^n + 0.5(v_j^{\perp} - |v_j^{\perp}|)\Psi_j^n, \tag{2}$$

where the advective normal velocity $v_j^{\perp}$ is evaluated at the face S_j, and assumes the following form:

$$v_j^{\perp} = \boldsymbol{S_j} \cdot 0.5[\boldsymbol{v_i} + \boldsymbol{v_j}]. \tag{3}$$

4 GPU Parallelization

The idea of GPU parallelization is based on decomposition of the MPDATA algorithm into blocks. Each block represents a part (submatrix) of all matrices, which are computed by one GPU task. Every task is a sequence of computational kernels which compute different parts of the algorithm.

In our approach, we distinguish the following levels of GPU parallelization:

- MPDATA task decomposition into kernels;
- overlapping of data transfer with computations;
- computations on GPU excecuted by GPU threads (work-items in OpenCL terminology).

Each MPDATA task is decomposed into 15 kernels, based on synchronization points and data dependencies. Each kernel computes a different part of MPDATA, and is configured in individual way considering the following OpenCL parameters:

- number of global work-items;
- number of local work-items;
- number of dimensions of work-group;
- vector size.

These kernels are executed in a FIFO order corresponding to the dependency tree expressing data dependencies between kernels. Fig. 1 shows the data dependency tree of MPDATA.

One of the most important feature of modern GPU architecture is possibility of overlapping data transfers with computations. It can be achieved by the stream processing. In our approach, each stream consists of a sequence of following instructions:

- sending data blocks from host memory to GPU global memory;
- computations performed by kernels;
- receiving data blocks from GPU global memory to host memory.

An example of stream processing on GPU that support overlapping of data transfers with computations is shown in Fig. 2. In the ideal case (with no time

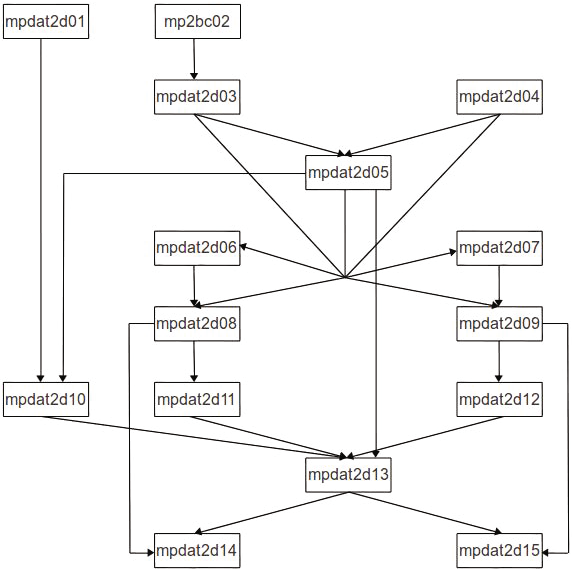


Fig. 1. Data dependency tree of MPDATA for GPU parallelization

delay of communication), we can observe that the more number of streams the more performance gaining can be achieved. Taking into account the time delay of communication, the maximum performance of our parallelization is achieved using four streams. This value was evaluated empirically.

Another level of parallelization are GPU threads called work-items. MPDATA is executed by work-items that are grouped in work-groups. In our approach, we use 1- or 2-dimensional work-groups. One of the biggest challenge here is providing the independence between work-groups because there is no synchronization mechanisms between them.

5 Performance Analysis for GPU Parallelization

The algorithm was tested on the NVIDIA Tesla C1060 card, ATI Radeon HD 5870 and NVIDIA Tesla M2070Q. Table 1 shows the performance results for the linear version of MPDATA with mesh of size 1024x1024, for 100 timesteps. As we can see, 70.3% of data transfer is overlapped with 38.7% of computations on C1060, while only 17.3% of data transfer is overlapped with computations on ATI. The overall time of MPDATA execution is shorter by 25% on ATI than on C1060, and by 29% on M2070Q than on ATI. The kernels time is shorter by 55% on ATI than on C1060, and by 14% on M2070Q than on ATI.

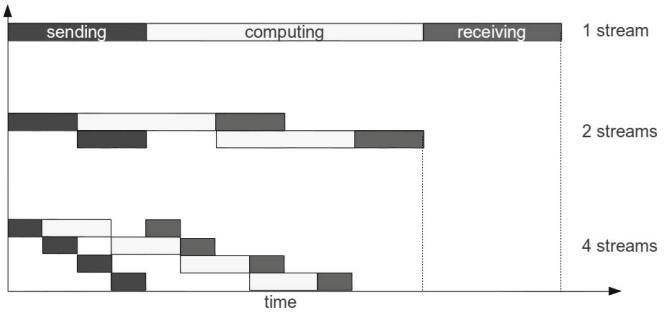


Fig. 2. Overlapping of data transfer with computations

Table 1. Performance results for GPU parallelization of linear version of MPDATA

	NVIDIA Tesla C1060			ATI Radeon HD 5870			NVIDIA Tesla M2070Q		
Streams count	1	2	4	1	2	4	1	2	4
Exec. time [s]	0.505	0.454	0.445	0.354	0.345	0.336	0.292	0.269	0.239
Kernels time [%]	63.6	63.2	64.5	40.8	44.8	69.8	43	43.4	46.5
Comm. time [%]	36.4	38.8	35.5	59.2	55.1	30.2	57	56.6	53.5
Overlapping kern. and comm. [%]	0	13.0	24.9	0	2.6	5.4	0	8.9	19.4
Kernel overlapped [%]	0	21.2	38.7	0	**4.7**	7.7	**0**	**20.3**	**41.6**
Communication overlapped [%]	**0**	**33.5**	**70.3**	**0**	4.2	**17.8**	0	15.6	36.2

6 CPU Parallelization

In this paper, when considering the CPU parallelization we focus on multi-core processing, vectorization and cache reusing. It is necessary to provide the load balancing of computations between available cores. For this aim, the whole problem is divided into six and four equal chunks for AMD and Intel CPUs, respectively. To achieve high efficiency of computations, it is required to apply the SIMD processing and provide a suitable data allocation in the main memory. The SIMD processing is applied manually, using the standard SSE and new AVX extensions. Aligning data to vector lengths is always recommended. When using SSE instructions, data should be aligned to 16 bytes. Similiarly, to achieve best results using Intel AVX instructions on 32-byte vectors, data should be aligned to 32 bytes. Therefore, each row of matrices is aligned to 16 and 32 bytes for SSE and AVX, respectively.

Fig. 3 shows the data dependency tree of MPDATA for the CPU implementation. The linear version of MPDATA corresponds to the first three stages marked

in Fig. 3 with f1, f2, and x', which conventionally are computed as the following sequence of steps:

```
f1: loading data; serial calculations; saving results;
f2: loading data; serial calculations; saving results;
x': loading data; serial calculations; saving results.
```

In order to exploit parallel features of CPU architecture, another approach is considered, which assumes the following steps:

```
f1: data partitioning; loading data; SIMD calc.; saving results;
f2: data partitioning; loading data; SIMD calc.; saving results;
x': data partitioning; loading data; SIMD calc.; saving results.
```

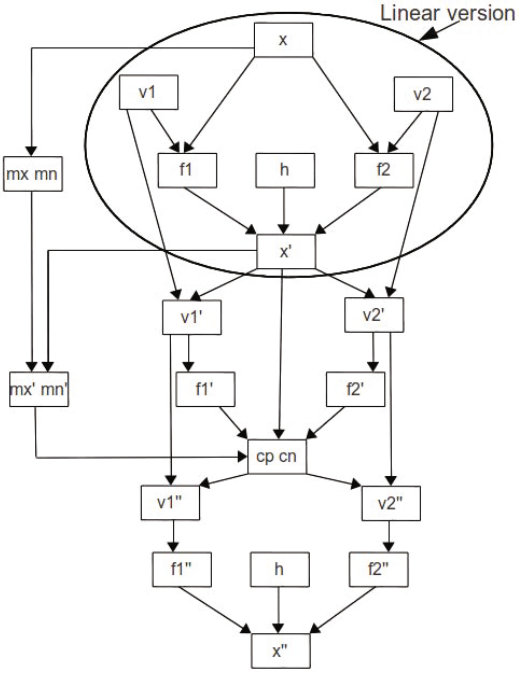


Fig. 3. Data dependency tree of MPDATA for CPU parallelizatoin

7 Performance Analysis for CPU Parallelization Using the Roofline Model

The algorithm was implemented using the OpenMP programming standard, as well as SSE and AVX extensions on Intel Xeon E3-1200 and AMD Phenom II X6. Table 2 shows the performance results of the linear version of MPDATA with mesh of size 5120x5120. As we can see, the speedup is only about 2, even

when using multicore and SIMD processing. In theory, the features of these architectures should allow for achieving maximum speedups of 32 and 24 for Intel and AMD CPUs, respectively. Therefore, we decided to use the Roofline Model [9] to identify bottlenecks of implementing MPDATA on these architectures.

Table 2. Performance results for standard approach of MPDATA

Mesh size 5120x5120	Intel Xeon E3-1200 (4 cores + AVX)		AMD Phenom II X6 (6 cores + SSE)	
	time [s]	speedup	time [s]	speedup
1 core without SIMD	0.16	-	0.21	-
multicore without SIMD	0.077	2.07	0.11	1.9
1 core with SIMD	0.079	2.02	0.16	1.3
multicore with SIMD	0.074	2.16	0.10	2.1

The Roofline Model shows inherent hardware limitations for a given kernel, as well as potential benefit and priority of optimizations. It relates processor performance to memory traffic. The model is based on the operational intensity parameter Q meaning the amount of operations per byte of DRAM traffic (flop/byte). The attainable performance R_a (flop/s) is then upper bounded by both the peak performance R_{max} (flop/s), and the product of the peak memory bandwidth B_{max} (byte/s), and the operational intensity Q:

$$R_a = min\{R_{max},\, B_{max} * Q\}\ \ [flop/s]\,. \tag{4}$$

Fig. 4 presents performance analysis for Intel Xeon E3 1270 CPU using the Roofline Model. For stages f1, f2 and x', the operation intensity can be expressed as:

$$Q = \frac{n \cdot m \cdot 25}{n \cdot l \cdot 4 \cdot 11} = 0.56\ [\frac{op}{byte}], \tag{5}$$

where computing a problem of size $n \times m$ requires $n \cdot m \cdot 25$ operations, and transfer of 11 matrices of size $n \cdot l \cdot 4$ bytes. Consequently, the attainable performance is only $R_a = 0.56\ [\frac{op}{byte}] \cdot 21\ [GB/s] = 11.7\ [Gop/s]$ as compared to 108.8 $[Gop/s]$ of peak performance.

The Roofline Model shows that the memory traffic is bottleneck when implementing MPDATA on multicore CPUs. To alleviate this limitation, we propose to use the loop tiling technique as a way of providing the efficient cache reusing. Thanks to that, partial results will be stored in cache, which reduces the memory bottleneck. This idea is presented in Fig. 5, where the size $nBlockSize \times mBlockSize$ of blocks has to be adjusted to the cache size.

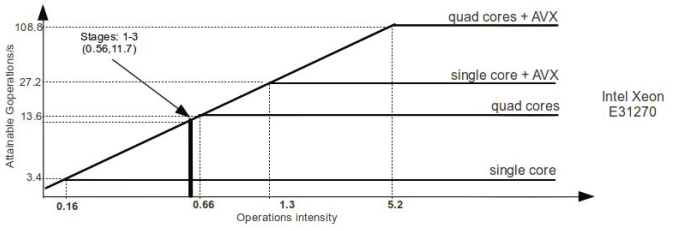


Fig. 4. The Roofline Model for Intel Xeon E3 1270

$for \frac{n}{nBlockSize} tiles \quad //i\ dimension$

$for \frac{l}{mBlockSize} tiles \quad //j\ dimension$

$MPDATA\_block(...)$ {
loading data from main memory to cache;
stage 1 : *parallel computations*;
saving partial results in cache;
stage 2 : *parallel computations*;
saving partial results in cache;
(...)
saving final results in main memory for each block;
}

Fig. 5. The idea of block version of MPDATA

8 Conclusions and Future Work

The heterogeneous GPGPU computing is a promising approach for increasing performance of numerical simulations of geophysical flows using the EULAG model. Our implementation supports multiple streams processing, which allows for overlapping data transfer with computations, as well as provides a significant reduction in the use of GPU global memory space. The MPDATA task decomposition allows for avoiding dependencies between work-groups. To achieve high efficiency of computations, it is required to apply the SIMD processing using dynamic size of vector.

The vector processing using the AVX extension allows for significant increase of CPU performance. The standard approach does not give a high performance. The obtained speedup is only about 2, even when using multicore and SIMD processing. Therefore, the performance analysis is provided. The Roofline Model shows that the memory traffic is bottleneck for the standard approach to MPDATA implemented on CPU. The loop tiling allows for efficient cache reusing to alleviate memory bottleneck. The block version of MPDATA requires

additional calculations for each block, however, this overhead can be reduced by storing partial results in cache.

Our parallelization of MPDATA is still under development. One of leading approaches is using the autotuning technique which allows for algorithm self-adapting to properties of a system architecture. The final result of our work will be adaptation of MPDATA to hybrid architectures with CPUs and GPUs. In this case, the first challenge is to provide high performance for each system's hybrid component, taking into account their properties. The second challenge concerns data partitioning and load balancing across heterogeneous resources.

Acknowledgments. This work was supported in part by the Polish Ministry of Science and Higher Education under Grant Nos. 648/N-COST/2010/0 COST IC0805 and BS/PB-1-112-3030/2011/S.

We gratefully acknowledge the help and support provided by Jamie Wilcox from Intel EMEA Technical Marketing HPC Lab. The authors are grateful to Krzysztof Luka from AMD for granting access to ATI Radeon HD 5870 GPU.

References

1. AMD Corporation: AMD Phenom II X6 Feature Summary, http://www.amd.com/
2. AMD Corporation: ATI Radeon HD 5870 Feature Summary, http://www.amd.com/
3. Smolarkiewicz, P.K.: Multidimensional positive definite advection transport algorithm: an overview. Int. J. Numer. Meth. Fluids 50, 1123–1144 (2006)
4. Kurowski, K., Kulczewski, M., Dobski, M.: Parallel and GPU based strategies for selected CFD and climate modeling models. Information Technologies in Environmental Engineering 3, 735–747 (2011)
5. Eulag Research Model for Geophysical Flows, http://www.eulag.com/
6. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro 28, 39–55 (2008)
7. Smolarkiewicz, P., Szmelter, J.: MPDATA: An edge-based unstructured-grid formulation. ELSEVIER Journal of Computational Physics 206, 624–649 (2005)
8. Gepner, P., Gamayunov, V., Fraser, D.L.: Early performance evaluation of AVX for HPC. ELSEVIER Procedia Computer Science 4, 452–460 (2011)
9. Williams, S., et al.: Roofline: an insightful visual performance model for multicore architectures. Communications of the ACM 52, 65–76 (2009)
10. Wyrzykowski, R., Rojek, K., Szustak, Ł.: Using Blue Gene/P and GPUs to Accelerate Computations in the EULAG Model. In: Lirkov, I. (ed.) LSSC 2011. LNCS, vol. 7116, pp. 670–677. Springer, Heidelberg (2012)

High-Resolution Simulation of Turbulent Collision of Cloud Droplets

Bogdan Rosa[1], Hossein Parishani[2], Orlando Ayala[2], Lian-Ping Wang[2], and Wojciech W. Grabowski[3]

[1] Institute of Meteorology and Water Management, National Research Institute, 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/
[3] Mesoscale and Microscale Meteorology Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307-3000, USA

Abstract. A novel parallel implementation of hybrid DNS (Direct Numerical Simulation) code for simulating collision-coalescence of aerodynamically interacting particles in a turbulent flow has been developed. An important application of this code is to quantify turbulent collision-coalescence rate of cloud droplets, relevant to warm rain formation, under physically realistic conditions. The code enables performing high-resolution DNS of turbulent collisions so the simulation results can be used to begin addressing the question of Reynolds number dependence of pair and collision statistics. The new implementation is based on MPI (Message Passing Interface) library, and thus the code can run on computers with distributed memory. This development enables to conduct hybrid DNS with flow field solved at grid resolutions up to 512^3 while simultaneously track up to several million aerodynamically-interacting droplets. In this paper we discuss key elements of the MPI implementation and present preliminary results from the high resolution simulations. The key conclusion is that, for small cloud droplets, the results on pair statistics and collision kernel appear to reach their saturation values as the flow Reynolds number is increased.

1 Introduction

Turbulent collision-coalescence of cloud droplets is a necessary step for warm rain initiation and development [19]. Rain drops are initiated and further grow in size primarily by colliding with cloud droplets that result from water vapor condensation on cloud condensation nuclei. The small size and small inertial response time of cloud droplets imply that pair statistics (i.e. radial distribution function RDF or droplet relative velocity) relevant to collision-coalescence are

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 401–410, 2012.

mainly determined by dissipation-range turbulence dynamics, making direct numerical simulation [2,17] a meaningful approach for this particular application. Due to the computational requirements, the domain size of DNS is typically in the range of 10 cm to 1 m scale only. Such a domain covers the flow dissipation range but an insufficient inertial subrange. As droplet size is increased, some inertial subrange scales of fluid motion can also contribute to the pair statistics. It is thus desirable to systematically increase the range of flow scales covered in DNS and consequently the computational domain size [1] in order to fully simulate the dynamic interactions of droplets and small-scale turbulence. Increase in domain size also implies an increase in the total number of simulated droplets. Furthermore, a long simulation or multiple realizations are often necessary to ensure a small statistical uncertainty of the computed pair statistics.

2 Methodology

Modeling of collision-coalescence of aerodynamically interacting droplets moving in a turbulent flow is a challenging task due to high computational cost and numerical and physical complexities. Dynamic and kinematic statistics of turbulent collision of cloud droplets depend primarily on the small-scale turbulent flow characteristics (e.g., Kolmogorov scales), settling velocity, and particle inertia. Statistical uncertainties of these physical quantities depend on the number of droplets followed in time and space. Assuming the dissipation-range flow is fully resolved, increasing the domain size translates to a higher flow Reynolds number, thus making simulations closer to physical conditions in turbulent clouds. A larger domain requires also tracking of a larger number of droplets, under the condition of a prescribed liquid water content (LWC).

The above considerations motivated us to utilize modern supercomputers with architecture based on distributed memory. The key question is how to take advantages of large computational resources (i.e. CPUs and memory) through efficient parallel implementation. Our hybrid DNS combines a pseudo-spectral simulation of turbulent air flow on a fixed spatial grid (Eulerian representation) with numerical integration of the equation of motion for freely moving droplets (Lagrangian representation). This combination of a large number of degrees of freedom on fixed grid points and a large number of degrees of freedom associated with moving droplets presents a significant challenge to efficient parallel implementation, as the two representations create different data structures that require different data distribution methods on available processors. Another challenge is that the pseudo-spectral method involves three-dimensional Fast Fourier transform (FFT) which requires global (i.e., whole-domain) data access or global data communication. These factors make the code communication-intensive.

Several different parallelization techniques could be considered. For the grid-based flow simulation, the most logical method is domain decomposition. For particle tracking, two different strategies can be considered: the first assigns to each processor a fixed subset of particles and their movements are handled by integrating their equations of motion; alternatively, each processor can treat

a fixed subset of particle pairs and particle-particle interaction forces. Some quantitative evaluation of the efficiencies of these two approaches is given in [12].

Here we present our MPI implementation based on 1D domain decomposition. This approach distributes evenly tasks associated with the computation of the turbulent flow field. The standard pseudo-spectral method is used to integrate the incompressible Navier-Stokes equations in a cubic domain with periodic boundary conditions. The Navier-Stokes equations are transformed into the spectral space where time advancement is performed. Parallel implementation of the 3D FFT based on 1D domain decomposition was developed in [6].

Several implementation issues related to the computation of particle trajectories have to be resolved. The first is the interpolation of the fluid velocity at the location of each particle, from the velocities at the fixed grid points. The second is the proper implementation of periodic boundary conditions for moving particles. The third is sending and receiving particles between neighboring subdomains when they cross the subdomain boundaries. Another is the detection of collision events. These aspects and their parallel implementations were carefully discussed in [13]. When local droplet-droplet aerodynamic interactions are considered [2], disturbance flow around each droplet has to be considered when advancing the location of other droplets within a given distance. This requires a significant data communication between neighboring domains [11].

3 Simulation Results

In this section, we present results obtained from our MPI code, including characteristics of the background turbulent flow field, and kinematic and dynamic statistics related to collision-coalescence of both aerodynamically-interacting and noninteracting droplets.

3.1 Background Turbulent Flow

First we present results of the simulated turbulent flows at different grid resolutions. Figure 1a shows Taylor microscale Reynolds number as a function of the grid resolution used. The Reynolds number is defined as $R_\lambda = \sqrt{15}u'^2\tau_k/\nu$ where u' is r.m.s. fluctuation velocity in a given direction, τ_k is Kolmogorov time and ν is fluid viscosity. We have utilized two different large-scale forcing schemes, namely, a stochastic forcing [15] and a deterministic forcing [16]. The purpose was to examine whether the collision and pair statistics are affected by the nature of large-scale forcing scheme. Using the deterministic forcing scheme we reached $R_\lambda = 205.2$ at grid size 512^3. Isihara et al. [10] and Donzis et al. [7] performed simulations with even larger Taylor microscale Reynolds numbers but their simulations were restricted to the single phase flow only. Figure 1b shows compensated energy spectra of the turbulent flows, using the stochastic forcing scheme, at several different mesh resolutions starting from 32^3 up to 1024^3. This plot demonstrates that the dissipation-range spectra completely overlap when $R_\lambda > 100$. The simulated flows at 512^3 and above also show that a portion of

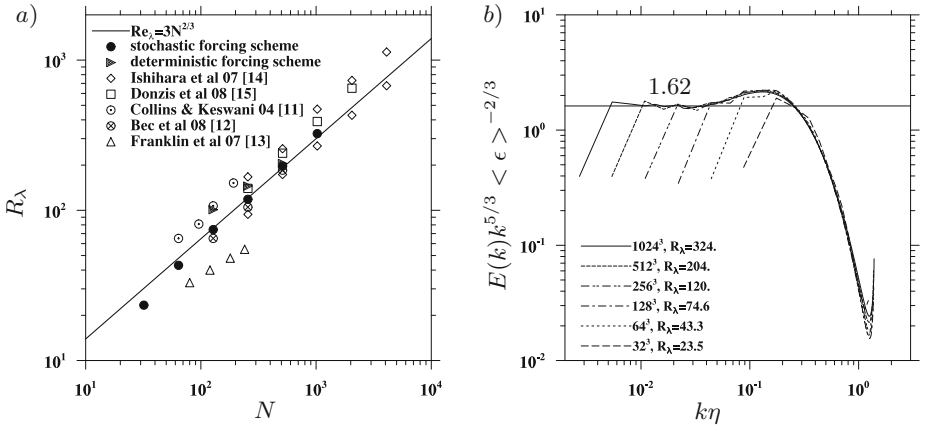


Fig. 1. *a*) Simulated flow Taylor microscale Reynolds number, R_λ, as a function of grid resolution and *b*) compensated energy spectra of the turbulent flows at different R_λ, using a stochastic large-scale forcing scheme. The line in *a*) indicates $R_\lambda = 3N^{2/3}$ [17].

the inertial sub-range scales are correctly represented, since the universal scaling (the line at 1.62) for the inertial subrange observed in experiments [9,14] is reproduced. Clearly the range of simulated flow scales increases with the flow Reynolds number (or grid resolution).

In Table 1, we list the flow statistics and indicate the computational domain size if a physical dissipation rate of 400 $\mathrm{cm^2 s^{-3}}$ is assumed in a turbulent cloud. Also listed is the number of droplets in the computational domain for a prescribed LWC at 1 $\mathrm{g/m^3}$, assuming that the system is bidisperse with half of the particles are 20 μm in radius and the other half 30 μm. These cloud droplet sizes are most relevant to warm rain initiation, as condensation and gravitational collision are both slow in growing these droplets and additional mechanism such as effects of turbulence could become critical. A larger number of droplets will reduce statistical uncertainty. In our previous OpenMP implementation, the total number of droplets in one single simulation might be limited by the amount of available memory. The MPI code can handle a larger number of particles ($\sim 10^7$) without such a technical restriction.

3.2 Parallel Performance

Parallel scalability of the new MPI implementation is evaluated using two modern supercomputers, *i.e.*, Lynx and Chimera. Lynx is a single cabinet Cray XT5m machine installed at National Center for Atmospheric Research (USA). The computer has seventy-six nodes, each with twelve processors, on two hex-core AMD 2.2 GHz Opteron chips. Chimera at the University of Delaware contains 3,168 cores which are grouped in 66 nodes. Each node has 4 CPUs AMD Opteron 6164HE 12-core running at 1.7GHz.

Table 1. Implications of increasing DNS grid resolutions. N is domain size (number of grid points in one direction), $< \epsilon >$ is the average energy dissipation rate in DNS units and u' is r.m.s. of fluid velocity.

N	R_λ	$< \epsilon >$	Domain size (cm) ($400\ \mathrm{cm}^2/\mathrm{s}^3$)	u'	Number of droplets LWC = $1\mathrm{g/m}^3$
Stochastic forcing scheme					
32	23.5	3646	6.0	7.08	1.0×10^3
64	43.3	3529	11.9	9.61	8.0×10^3
128	74.6	3589	23.9	12.61	6.6×10^4
256	120.	3690	48.1	16.18	5.4×10^5
512	204.	3900	97.5	20.84	4.5×10^6
1024	324.	3777	193.0	26.29	3.5×10^7
Deterministic forcing scheme					
256	144.9	0.2011	26.32	17.56	2.6×10^5
512	205.2	0.2146	45.02	20.90	1.2×10^6

In the first test, the influence of compiler options on multi-CPU performance of our MPI code is investigated. The runs were conducted on Lynx employing two popular compilers, PGI and Intel. For each series of runs, all components of the code were compiled with the highest optimization level. The simulations employed a grid resolution of 512^3 and 5 million of non-interacting particles. The total number of processors was fixed to 64, but the number of processes per node changed between runs. Figure 2a shows the average wall clock time per time step from each run. The wall clock times for the flow solver and for tracking particles are shown separately. We conclude that the total wallclock time with the Intel compiler is shorter by 6 do 12 % than that with the PGI compiler. The difference also depends on the distribution of processes between nodes and is mainly related to particle tracking. Additionally, Fig. 2a shows that the parallel efficiency drops (by more than 50%) as the number of processes per node is increased, due to memory bus saturation in this multi-core system.

In the second test, the speedup factors of the MPI implementation on two different machines are examined. The same testing problem is used, namely, 512^3 flow grid and 5 million particles. The runs on Lynx were performed for two different numbers of processes per node, *i.e.*, 12 which uses the least number of nodes, and 1 which corresponds to the best performance. The run with 128 processors on Lynx was done with 2 processes per node, to insufficient number of nodes. The runs employing all 12 processors per node were limited to 32 processes due to the memory constraint. Figure 2b displays results from both Lynx and Chimera. Also shown are the scalability data of the previous OpenMP implementation conducted on the IBM Power 575 cluster (4064 POWER6 processors running at 4.7 GHz) at NCAR.

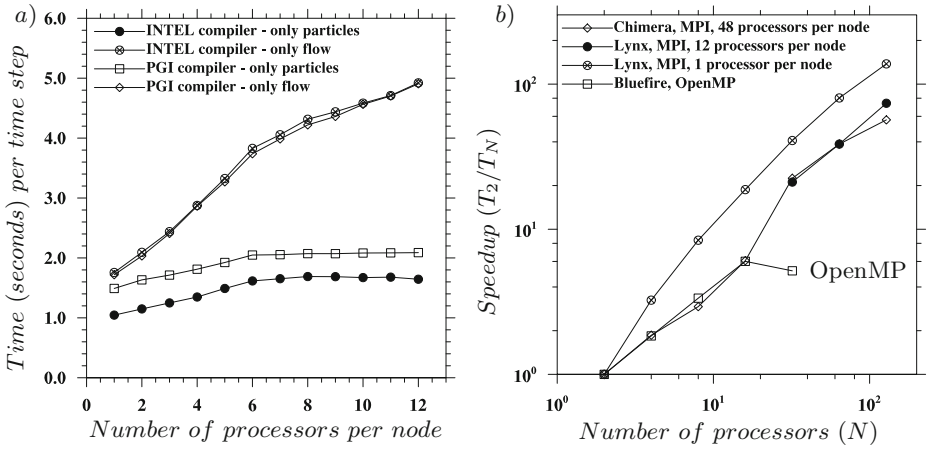


Fig. 2. Scalability data of the MPI code for simulations at 512^3 grid size with 5 million (non-interacting) particles. Panel *a*): wall clock time obtained with two different compilers on Lynx as a function of the number of processors per node (the total number of processes is fixed to 64). Panel *b*): speedup factors as a function of the number of processors. Also speedup data of previous OpenMP implementation performed on NCAR's Bluefire are plotted.

3.3 Kinematic and Dynamic Statistics for Droplets

Figure 3 shows single-droplet r.m.s. fluctuation velocities in horizontal *a*) and vertical *b*) direction, respectively. Thick lines represent the theoretical prediction from [3,18]. For vertical direction the theoretical prediction have been developed following the procedure from [3] for the horizontal direction.

For most cases, the droplet r.m.s velocities approach the corresponding fluid r.m.s. velocity as the droplet radius is reduced. The horizontal droplet r.m.s velocity drops more quickly with droplet size than the vertical velocity due to the sedimentation [18]. This difference becomes more evident at larger flow Reynolds numbers. Here the key point is that the single-droplet r.m.s. fluctuation velocity increases with flow Reynolds number or simulation domain size as the fluid r.m.s. velocity increases when more larger scales are included. The usual Reynolds number scaling is $u' \approx 0.5 v_k R_\lambda^{0.5}$ so the single-droplet r.m.s. velocity increases monotonically with R_λ.

Droplet-droplet pair statistics, such as radial distribution function (RDF) [20] and radial relative velocity are kinematic parameters directely proportional collision rate. The monodisperse pair statistics of nearly touching particles are shown in Figs. 4a and 4b, along with results at other flow Reynolds numbers. For larger droplets, the pair statistics increase with the flow Reynolds number. However, for any given droplet size, there is a tendency of saturation, namely, the pair statistics eventually become insensitive to flow Reynolds number. This saturation for smaller droplets is reached at smaller flow Reynolds number, since the range of flow

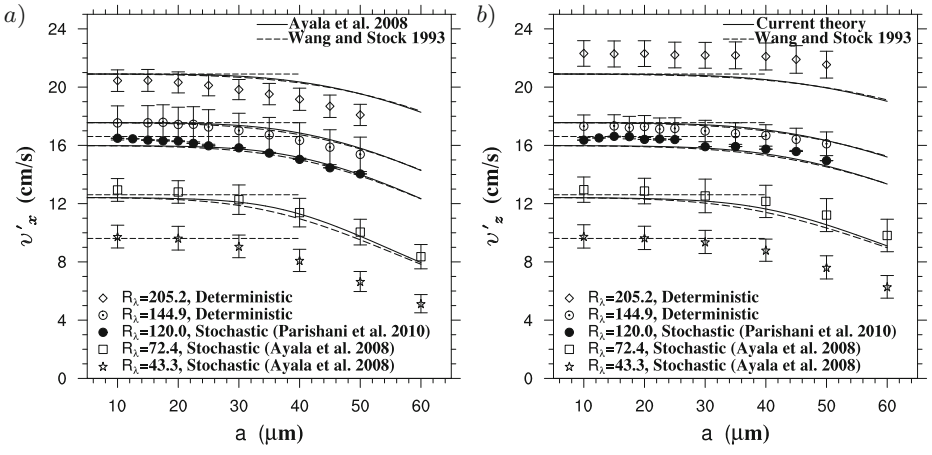


Fig. 3. Single-droplet r.m.s. fluctuation velocity in *a*) horizontal and *b*) vertical direction. Dashed thin lines mark r.m.s. fluctuation velocity of the fluid. Thicker lines represent the theoretical prediction from [3,18].

scales affecting the pair statistics is more limited for smaller droplets due to their smaller Stokes number. For example, saturation is observed to occur at $R_\lambda \approx 200$ for 30 μm droplets. The gradual saturation of pair and collision statistics with R_λ is further demonstrated in Fig. 5, where the statistics are plotted as a function of R_λ. The dynamic collision kernel and RDF of the mondisperse system both show evidence of saturation. This observed saturation justifies the hybrid DNS approach using flow Reynolds numbers significantly less than those in real clouds.

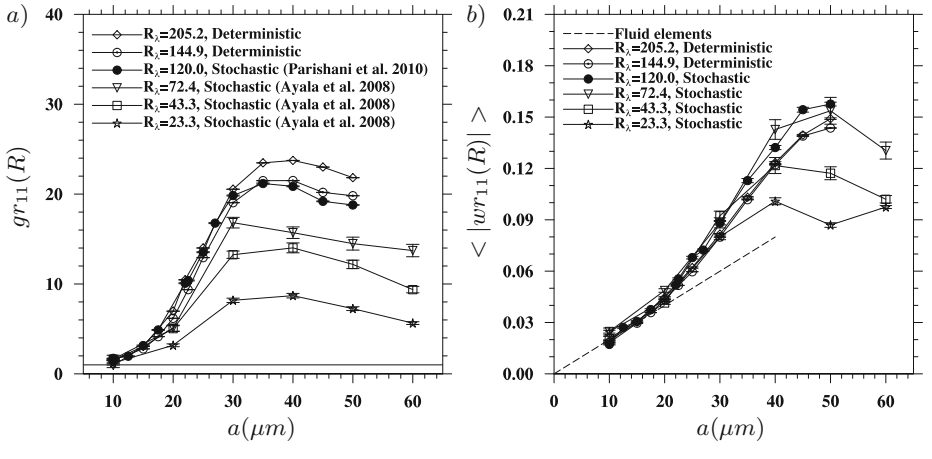


Fig. 4. *a*) Radial distribution function and *b*) relative velocity at contact $r = R$ for monodisperse pairs as a function of droplet size for different values of R_λ

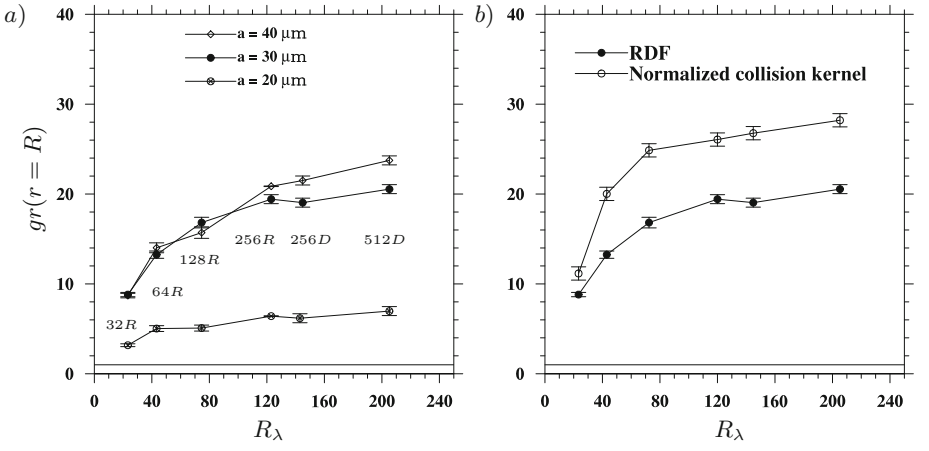


Fig. 5. *a*) Monodisperse RDF as a function of R_λ for three droplet sizes, *b*) dynamic collision kernel of the mondisperse system (30μm) for different R_λ normalized by $2\pi R v_{GRAV}$ where v_{GRAV} is terminal velocity of the particles in the stagnant flow

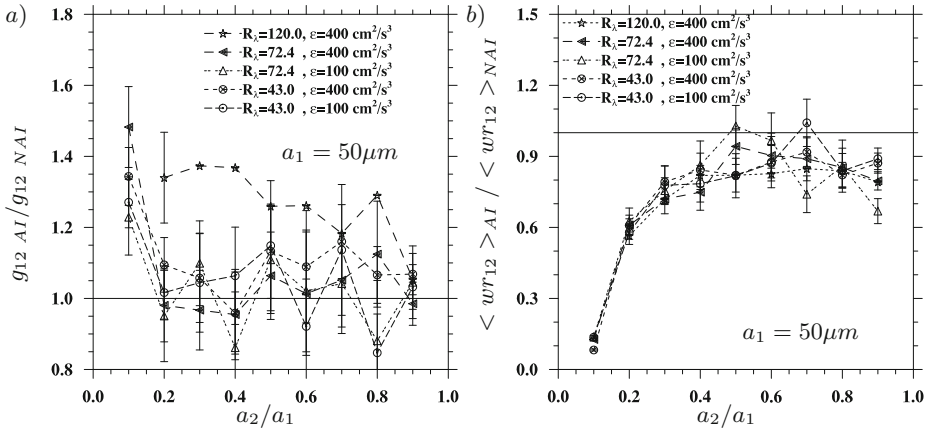


Fig. 6. Ratio of *a*) the radial distribution function and *b*) the radial relative velocity with AI to that corresponding to geometric collisions

Finally, we show pair statistics for aerodynamically interacting (AI) droplets [2] in Fig. 6. The results are normalized by corresponding values without aerodynamic interactions (no-AI). Here we simulate bidisperse system where one set of droplets is 50 μm in radius. The results show that the aerodynamic interaction has a strong influence on the kinematic statistics for the pairs of very different sizes, leading to a large reduction of $< w_{r,12} >$ and a significant increase in g_{12}. In this case, the disturbance flow of the larger droplet significantly alters the trajectory of the smaller droplet. When the pair are similar in size, the impact

of aerodynamic interaction is relatively weak. While it is difficult to infer the precise effect of R_λ due to large statistical uncertainty, we may conclude that the flow Reynolds number effect is relatively weak for the cases considered here.

4 Summary and Conclusions

In this paper, we have reviewed a hybrid DNS approach designed to study turbulent collision-coalescence of cloud droplets. A general question is whether such an approach is justified for this application problem. As a first step to address this question, we have implemented MPI in order to increase the range of flow scales that can be simulated. We reviewed briefly various MPI implementation issues associated with the turbulent flow simulation and with tracking the motion and detecting pairwise interaction of droplets.

The MPI code is then used to simulate the flow and dynamics of droplets at higher flow Reynolds numbers, using up to 1024^3 grid for the flow only simulations and up to 512^3 grid for flows laden with droplets. The results show that the inertial subrange of turbulence can be correctly reproduced at high grid resolutions. As the flow Reynolds number or the computational domain size is increased, the range of flow scales is also increased, leading to increased single-droplet r.m.s. fluctuation velocities. However, we show that the pair statistics and the dynamic collision kernel will reach their saturated values, at least to the leading order in R_λ, if all relevant scales of fluid motion are included in the flow simulation. This supports a fundamental assumption in the hybrid DNS, namely, hybrid DNS at much lower flow Reynolds numbers compared to those in real clouds can be used to quantity turbulent collision-coalescence of cloud droplets. The R_λ dependence of pair and collision statistics found in previous low-resolution simulations is a result of inadequate flow scale separation.

The MPI code is currently based on one-dimensional spatial domain decomposition. We are currently extending the MPI implementation using two-dimensional spatial domain decomposition, so that a much larger number of distributed-memory processors can be used, to further increase the flow Reynolds number or computational domain size. The ultimate goal is that all relevant scales of small-scale turbulence can be simulated so the pair statistics and pairwise interactions of cloud droplets of radii 100 μm or less can be fully studied.

Acknowledgements. This work was supported by the National Science Foundation (NSF) under grants ATM-0527140, ATM-0730766, OCI-0904534, and CRI-0958512. Computing resources are provided by National Center for Atmospheric Research (NCAR CISL-35751010, CISL-35751014, and CISL-35751015).

References

1. Ayala, O., Rosa, B., Wang, L.P., Grabowski, W.: Effects of turbulence on the geometric collision rate of sedimenting droplets: Part 1. results from direct numerical simulation. New J. Physics 10(075015) (2008)

2. Ayala, O., Grabowski, W.W., Wang, L.P.: A hybrid approach for simulating turbulent collisions of hydrodynamically-interacting particles. JCP 225
3. 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)
4. Bec, J., Biferale, L., Bofftta, G., Celani, A., Cencini, M., Lanotte, A., Musacchio, S., Toschi, F.: Acceleration statistics of heavy particles in turbulence. Journal of Fluid Mechanics 550, 349–358 (2006)
5. Collins, L.R., Keswani, A.: Reynolds number scaling of particle clustering in turbulent aerosols. New Journal of Physics 6(119) (2004)
6. 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)
7. Donzis, D., Yeung, P., Sreenivasan, K.: Dissipation and enstrophy in isotropic turbulence: resolution effects and scaling in direct numerical simulations. Phys. Fluids 20(045108) (2008)
8. 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)
9. Ishihara, T., Gotoh, T., Kaneda, Y.: Study of high-reynolds number isotropic turbulence by direct numerical simulation. Annual Review of Fluid Mechanics 41, 165–180 (2009)
10. Ishihara, T., Kaneda, Y., Yokokawa, M., Itakura, K., Uno, A.: Small-scale statistics in high-resolution direct numerical simulation of turbulence: Reynolds number dependence of one-point velocity gradient statistics. J. Fluid Mech., 335–366 (2007)
11. Parishani, H., Rosa, B., Grabowski, W.W., Wang, L.P.: Towards high-resolution simulation of turbulent collision of cloud droplets. In: Proceedings of the 7th International Conference on Multiphase Flow, Tampa, Fl, USA (2010)
12. Plimpton, S.: Fast parallel algorithms for short-range molecular-dynamics. Journal of Computational Physics 117(1), 1–19 (1995)
13. Rosa, B., Wang, L.P.: Parallel Implementation of Particle Tracking and Collision in a Turbulent Flow. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009. LNCS, vol. 6068, pp. 388–397. Springer, Heidelberg (2010)
14. Sreenivasan, K.: On the universality of the Kolmogorov constant. Physics of Fluids 7, 2778–2784 (1995)
15. 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)
16. Wang, L.P., Rosa, B.: A spurious evolution of turbulence originated from round-off error in pseudo-spectral simulation. Computer and Fluids 38, 1943–1949 (2009)
17. Wang, L.P., Rosa, B., Gao, H., He, G., Jin, G.: Turbulent collision of inertial particles: point-particle based, hybrid simulations and beyond. Int. J. Multiphase Flow 35, 854–867 (2009)
18. Wang, L.P., Stock, D.: Dispersion of heavy particles by turbulent motion. J. Atmos. Sci. 50(13) (1993)
19. Wang, L.P., Xue, Y., Ayala, O., Grabowski, W.: Effects of stochastic coalescence and air turbulence on the size distribution of cloud droplets. Atmospheric Research 82, 416–432 (2006)
20. 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)

Parallelization of the Seismic Ray Trace Algorithm

Kamil Szostek and Andrzej Leśniak

AGH University of Science and Technology,
Faculty of Geology, Geophysics and Environment Protection,
Department of Geoinformatics and Applied Computer Science,
al. A. Mickiewicza 30, 30-059 Kraków, Poland
{szostek,lesniak}@agh.edu.pl

Abstract. This article presents the parallelization of seismic ray trace algorithm. The chosen Urdaneta's algorithm is shortly described. It provides wavelength dependent smoothing and frequency dependant scattering thanks to the implementation of Lomax's method for approximating broad-band wave propagation. It also includes Vinje *et al.* wavefront propagation technique that provides fairly constant density of rays. Then the parallelized algorithm is preliminarily tested on synthetic data and the results are presented and discussed.

Keywords: raytrace, 2D seismic modeling, parallel algorithm.

1 Introduction

Ray trace algorithms are the techniques widely used in computer calculation and simulation, whenever tracing rays needs to be performed. In computer graphics, one of the most popular ray trace problems, these algorithms are implemented to achieve high quality images of virtual scenes, including reflections, refractions, shadows and more. Also in medicine ray trace algorithms are used to achieve extremely high quality images in simulation or reconstruction [1]. In seismology, the accuracy of ray trace algorithms is essential in successful seismic events localization and inversion of seismic source parameters [2]. Furthermore, in seismic modeling different ray trace algorithms have evolved during last years to satisfy computation time and accuracy needs. Eikonal equation solving technique has been developed in parallel systems and presented in [3]. Different approach showed in [4] adapt staggered-grid, finite-difference scheme. Another promising raytracing algorithm was introduced by Asakawa in [5] with later improvement by W. Hao-quan [6].

Algorithms parallelization is one of the most important techniques to improve computational speed. The development of multi-cored processors, highly efficient graphic cards and cluster technologies offers programmers and computer scientists significant acceleration of their algorithms with minimum costs [7]. Urdaneta presented sophisticated join of two techniques and his solution will be briefly presented in this paper and analyzed in terms of calculation speed and parallelization potential.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 411–418, 2012.

2 Urdaneta's Algorithm

2.1 Wavefront and Waverays

Lomax presented a new method for approximating broad-band wave propagation in complex velocity models.[8] This approximation is done using Gaussian weight curve (Fig. 1a). Then the Huygen's principle provides wavepath bending $\Delta\hat{s}$ from points $\boldsymbol{x}_v^{-1}$ and $\boldsymbol{x}_v^{1}$ (Fig. 1b). Advantages of this technique are: wavelength dependent smoothing which increases wavepaths stability, frequency dependant scattering and capability of handling large to small inhomogeneity sizes (Fig. 2).

Urdaneta in [9] has joined together Lomax's waverays approximation method with Vinje *et al.* [10] wavefront method. Wavefront is defined as a curve with equal traveltime. Its construction presented by Vinje *et al.* implements different

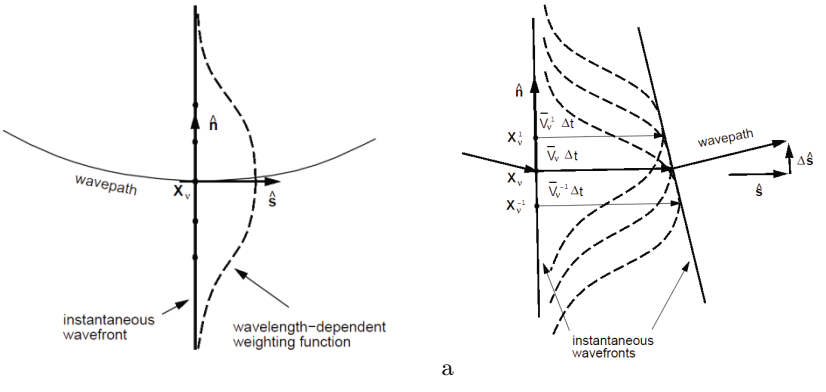


Fig. 1. Lomax's wavelength-dependent weighting function is evaluated by averaging velocities with Gaussian curve centered at point $\boldsymbol{x}_v$[9]

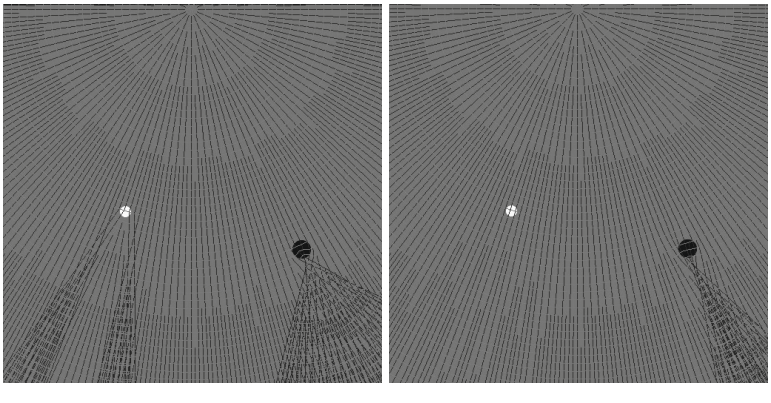


Fig. 2. Lomax's raytracing is wavelength and frequency dependant: on the left frequency was set to 500Hz, on the right to 10Hz. Rays are influenced by small inhomogeneities. Here they are also interpolated using Vinje *et al.* method.

approach to parameters computation: instead of calculating each ray separately, traveltimes and amplitudes are obtained from constructed wavefront. Propagation in medium is achieved by performing raytrace on preceding wavefront (Fig. 3a). If the distance between two successive rays is greater than predefined *DSmax*, a new ray is interpolated between two old ones. This interpolation is done using third order polynomial. Grid values are calculated from surrounding waverays and wavefront (Fig. 3b). New amplitudes are calculated in a similar way.

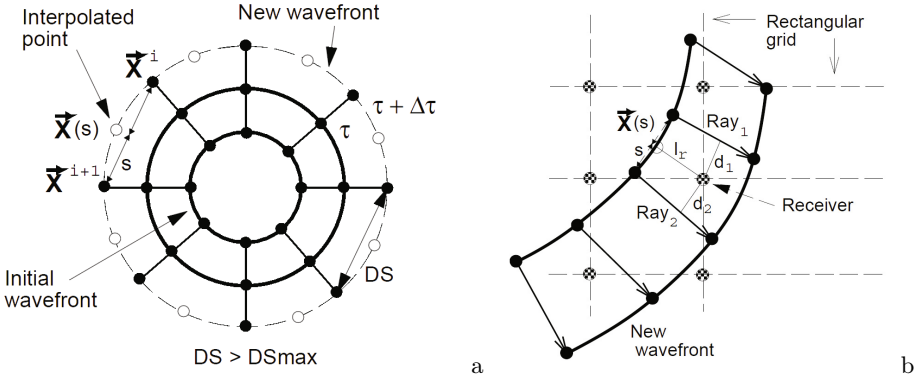


Fig. 3. Wavefront construction presented by Vinje *et al.* Points at time $\tau + \Delta\tau$ are calculated from previous wavefront. If distance between new points is greater than predefined *DSmax* a new point $\boldsymbol{x}_s$ is interpolated (a). Traveltimes and amplitudes are interpolated onto the grid as presented in figure b [9].

2.2 Urdaneta's Algorithm

Urdaneta's final algorithm begins with the definition of structure of ray in the wavefront. This structure, called *cube*, is as followed:

- points *x0* and *x1* – positions of the beginning and the end of the ray, respectively,
- *angle* – arriving angle of the ray at point *x1*,
- *ampl* – amplitude at the ray end,
- *cf* – flag that describes the ray status.

The maximum size of the wavefront is predefined. In case of exceeding this value, an error message is sent and the algorithm stops. In the initialization function, the first wavefront is defined – the 'shot point' – with starting points *x0* and proper angles. The propagation of wavefront is performed using Lomax's algorithm for waverays. The wave frequency and timestep are arbitrarly chosen. For each ray in the wavefront array, a new position (point *x1*) and arriving angle are calculated. Then, if there is self crossing wavefront or any ray is out of borders, status flag is properly set. In next step, these rays are removed from the array. Amplitudes, *gridding* and new rays interpolation are done using Vinje

et al. methods. Finally, the wavefront proceeds to next timestep. It is complete by replacing point *x0* with *x1*. These calculations repeat until wavefront array size is more than 4 (Fig. 4).

In order to calculate all arrivals times, *selfcrossing check* subroutine should be deactivated, but, for the sake of simplicity and to reduce the computational time, crossed rays are removed.

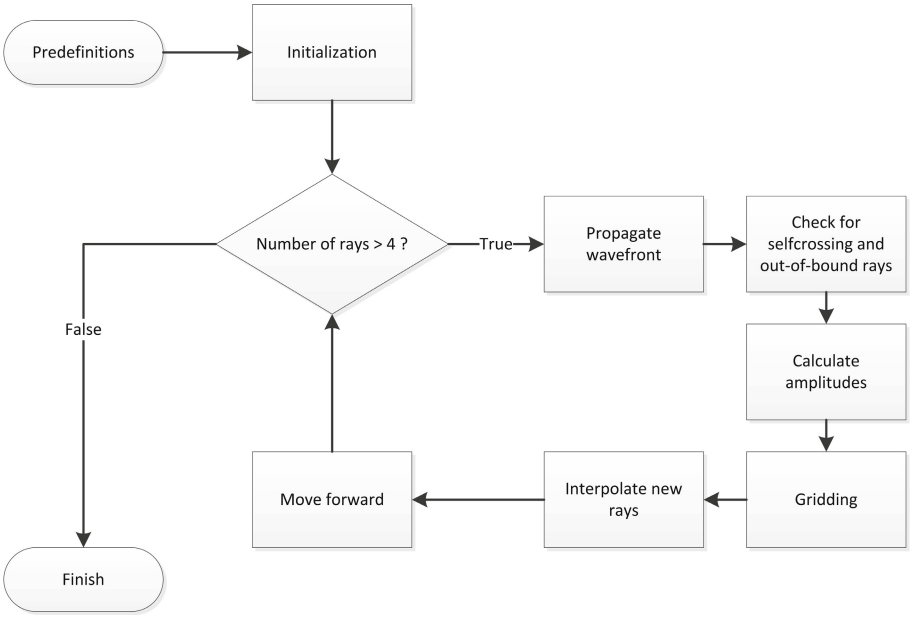


Fig. 4. Diagram of Urdaneta's algorithm [9]

3 Algorithm Parallelization

3.1 CPU Utilization

Urdaneta's algorithm has been implemented in C# (.NET 4.0) and tested using synthetic velocity models. Before any parallelization, CPU workload should be measured to get information, which procedures have the largest influence to the computation speed. The ANTS Performance Profiler was used to obtain CPU most time consuming function. Table 1 shows the results of this analysis. Insignificant methods are omitted. The profiling and later tests was performed using 512m x 512m synthetic isotropic velocity model, with 1600 grid nodes and timestep equal 3ms. Shot-point was set to the x=256m and z=256m.

The most time consuming procedure, according to profiler results, is gridding. It takes about 80% of whole computation time for chosen set of parameters. This procedure loops over every grid point on every wavefront step, so the size of the grid has a major influence on raytrace computation time. Enlarging the number of nodes will result in increasing gridding computation time.

Table 1. Results of Urdaneta's algorithm time analysis with ANTS Performance Profiler

Procedure Name	CPU Time [%]	CPU Time With Children[%]	Hit Count
RaytraceTest	0.029	97.851	1
Gridding	76.964	79.811	56
PropagateWavefront	0.009	5.204	56
CalculatePropagation	0.079	5.173	5581
GetAvarageVelocity	2.345	5.044	16743
CheckForSelfCrossing	1.734	3.480	55
GetDistance	2.862	2.862	3648859
CheckIntersection	1.715	1.715	371403
CalculateAmplitudes	0.240	0.240	56

3.2 Parallelization Procedure

The major idea is to change the most time consuming process, *gridding*, into its parallelized version by domain decomposition of the nodes grid. The *gridding* procedure consists of loop over all grid nodes and this loop can be parallelized, which means that grid nodes can be calculated simultaneously, e.g. in separated threads. As Table 1 shows, *gridding* procedure was called 56 times during profiling – each call for each of wavefronts. Therefore *gridding* procedure is moved outside the main loop. Tests have showed that this can decrease the computation time by about 10%. *Gridding* function is then performed with another loop over all calculated wavefronts, which are stored in array during wavefront propagation (Fig. 5).

The Parallelization is made by creating a thread pool for each processor found and performing *gridding* partially in each thread. Each processor works on part of the grid and calculates nodes values using Vinje *et al.* interpolation method (Fig. 3b). This solution is possible, because there is no interaction between already calculated wavefronts as well as between grid nodes.

3.3 Results of Preliminary Test

Parallelized algorithm was preliminarily tested on dual core processor. Synthetic tests were made for different grid size, starting with 10x10 grid nodes up to 500x500. Results of tests are presented in Table 2.

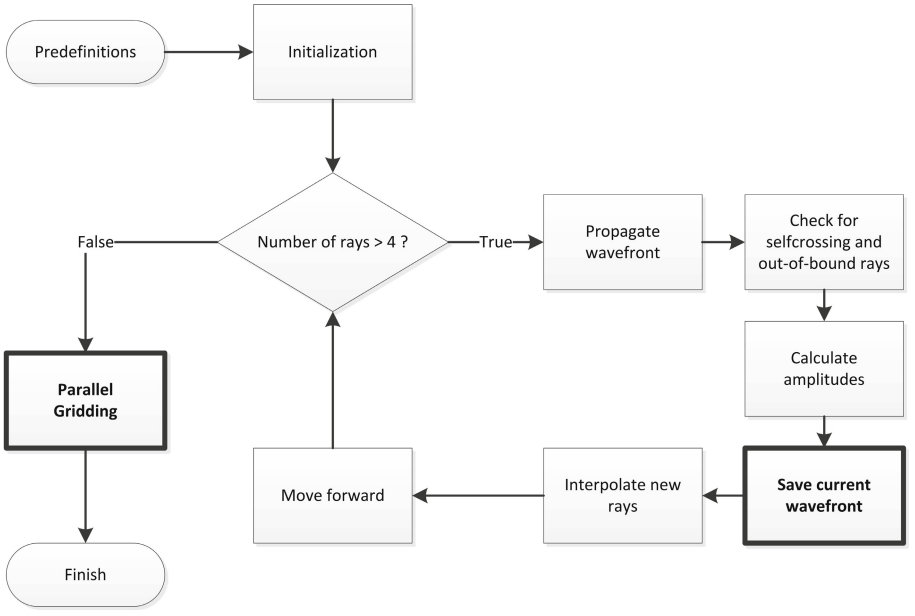


Fig. 5. Diagram of modified Urdaneta's algorithm with parallelization

Table 2. Preliminary test results

Grid size	*gridding* part [%]	Computation time [ms]		Acceleration
		1 CPU	2 CPUs	
10x10	44.09	156.00	140.40	1.111
20x20	76.51	358.80	249.60	1.438
30x30	87.80	702.00	436.80	1.607
40x40	92.63	1170.00	686.40	1.705
50x50	94.46	1778.40	998.40	1.781
60x60	96.44	2527.20	1388.40	1.820
70x70	96.88	3416.41	1872.00	1.825
80x80	97.58	4399.21	2418.00	1.819
90x90	98.25	5506.81	3010.81	1.829
100x100	98.79	6817.21	3697.21	1.844
150x150	99.47	15319.23	8174.41	1.874
200x200	99.63	26988.05	14414.43	1.872
250x250	99.82	42510.07	22604.44	1.881
300x300	99.85	60886.91	32370.06	1.881
350x350	99.87	83116.95	43960.88	1.891
400x400	99.90	108154.99	57486.10	1.881
450x450	99.94	136859.04	72680.53	1.883
500x500	99.93	169260.30	89996.56	1.881

The tests proved that parallelization was done correctly. For each grid size the acceleration is more than 1.85, except the small ones: small grid sizes are calculated faster and parallelizable part takes less time, as presented in Table 2. What is more, usually small grid sizes are considered as less relevant.

4 Summary and Future Work

This paper is an introduction to the subject of ray trace algorithms parallelization, presenting the parallelization of one of the seismic ray trace algorithm. It computes first arrival travel-times and amplitudes estimation in 2D medium using Lomax's ray trace method and Vinje *et al* wavefront propagation technique, both joined by Urdaneta. Parallelization has been done on the most time consuming procedure, *gridding*, which has the most influence on the algorithm speed. Preliminary tests on dual core CPU proved that acceleration is considerably high for relatively dense grids of nodes. Future work will be done in several areas of interest. Firstly, the presented parallel algorithm is going to be tested on machines with more than two CPUs to prove its reliability. Secondly, presented technique will be compared to other popular travel time and amplitude estimation techniques and their parallel versions in the matter of computation time and errors. What is more, as the GPU calculations become more popular and efficient, presented ray trace algorithm will be implemented and tested on graphic hardware, also in context of server side calculations [11].

Acknowledgements. The study was financed from the statutory research project No 11.11.140.032 of the Department of Geoinformatics and Applied Computer Science, AGH University of Science and Technology.

References

1. Hadwiger, M., Ljung, P., Rezk Salama, C., Ropinski, T.: Advanced illumination techniques for GPU-based volume raycasting. In: ACM SIGGRAPH 2009 Courses (SIGGRAPH 2009), pp. 1–166. ACM, New York (2009)
2. Scales, J.: Theory of seismic imaging. Samizdat Press (1994), http://samizdat.mines.edu/imaging/
3. Jeong, W.-K., Whitaker, R.T.: A fast eikonal equation solver for parallel systems. Imaging 84112, 1–4 (2007)
4. Shunhua, C., Greenhalgh, S.: Finite-difference solution of the eikonal equation using an efficient, first-arrival, wavefront tracking scheme. Geophysics 59(4), 632–643 (1994)
5. Asakawa, E., Kawanaka, T.: Seismic raytracing using linear traveltime interpolation. Geophysical Prospecting 41(1), 99–111 (1993)
6. Hao-quan, W.: An Improved Method of Linear Travel-Time Interpolation Ray Tracing Algorithm. Acta Physica Polonica A 118(4), 521–526 (2010)
7. Danek, T.: Parallel computing and PC clusters in seismic wave field modeling. Geoinformatica Polonica 7, 25–34 (2005)

8. Lomax, A.: The wavelength-smoothing method for approximating broad-band wave propagation through complicated velocity structures. Geophys. J. Int. 117(2), 313–334 (1994)
9. Urdaneta, H.: Wavefront construction using waverays. Stanford Exploration Project, Report 80, 85–100 (2001)
10. Vinje, V., Iverson, E., Gjøystdal, H.: Traveltime and amplitude estimation using wavefront construction. Geophysics 58(8), 1157–1166 (1993)
11. Szostek, K., Piórkowski, A.: OpenGL in Multi-User Web-Based Applications. In: Innovations in Computing Sciences and Software Engineering, SCSS 2009, pp. 379–383. Springer, Heidelberg (2010)

A Study on Parallel Performance of the EULAG F90/95 Code

Damian K. Wójcik, Marcin J. Kurowski, Bogdan Rosa,
and Michał Z. Ziemiański

Institute of Meteorology and Water Management
National Research Institute,
ul. Podleśna 61, 01-673 Warsaw, Poland
damian.wojcik@imgw.pl

Abstract. The paper presents several aspects of the computational performance of the EULAG F90/95 code, originally written in Fortran 77. EULAG is a well-established research fluid solver characterized by robust numerics. It is suitable for a wide range of scales of the geophysical flows and is considered as a prospective dynamical core of a future weather forecast model of the COSMO consortium. The code parallelization is based on Message Passing Interface (MPI) communication protocol. In the paper, the numerical model's parallel performance is examined using an idealized test case that involves a warm precipitating thermal developing over an undulated terrain. Also the efficiency of the basic code structures/subroutines is tested separately. It includes advection, elliptic pressure solver, preconditioner, Laplace equation solver and moist thermodynamics. In addition, the effects of horizontal domain decomposition and of the choice of machine precision on the computational efficiency are analyzed.

1 Introduction

Numerical Weather Prediction (NWP) is based on mathematical models of fluid dynamics, describing physical processes governing atmospheric flows. In practice, partial differential equations of the model are solved numerically in a 3-dimensional space and time. What makes the NWP extremely complex is the range of spatial scales involved. It spans from the microscale of turbulence and in-cloud process ($O(10^{-2}$ m)) to global scale of planetary flows ($O(10^{7}$ m)). One of the consequences is the essential role of computer sciences in a development of numerical tools for timely and reliable weather forecasting.

First, the need of realistic representation of weather processes requires possibly high resolution of model's numerical grid. On the other hand, any useful operational forecast has to be calculated in times much shorter than a prognostic timescale. In practice, a forecast for about a week should be obtained within a few hours. Therefore, a weather forecast model needs both possibly large computer resources and robust and efficient numerical code to optimize

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 419–428, 2012.

these constrains. It is worth mentioning that with the continuous progress in available computer resources contemporary regional weather models reach horizontal resolution of O(10^3 m) which already allows for explicit representation of convective processes.

The paper reports on work aimed at testing parallel performance of the EULAG F90/95 code originally written in Fortran 77. The project is developed within the frame of the "Conservative Dynamical Core" priority project of the COSMO consortium. The project is focused on implementation of anelastic non-hydrostatic model EULAG (Prusa et al. [1]) as a future high-resolution dynamical core of the COSMO model (see [2]). The preliminary results were presented in Piotrowski et al. [3]. The current paper focuses on the key problems of the computational efficiency and scalability of the EULAG code. Basic discussion concerning EULAG scalability and computational performance was presented by Prusa et al. [1] for numerical model configuration involving Held-Suarez global climate (Held and Suarez [4]) and solar magneto-hydrodynamic test cases. Here, the focus is on a performance of main EULAG procedures for a simplified weather-like application of warm precipitating thermal rising above a hill in a stably stratified atmosphere. It allows more realistic verification of EULAG in a configuration employing curvilinear framework with phase-change processes included. Additional aspects considered are the influence of machine precision and the type of domain decomposition on the efficiency of the code.

2 Benchmark Experiment

2.1 Experiment Setup

The efficiency of the EULAG parallel code was tested using an idealized three-dimensional benchmark experiment of a warm bubble developing in moist, stable and initially motionless atmosphere. Instead of Cartesian coordinates, a more general curvilinear framework employing terrain-following coordinates (Gal-Chen and Somerville [5]), typical for NWP applications, was imposed over a cosine-like orography (see Fig. 1). The numerical experiments differ by grid number in horizontal dimensions of the computational domain, while vertical grid number remains the same. The grid number changes from $32\times32\times64$ to $1024\times512\times64$ grid points (*i.e.* length $\times$ width $\times$ height) while the physical domain has a fixed size of $1000\times1000\times1000$ meters. Total integration time is always 200 seconds and the integration step is 1 s. The radius of the warm air bubble is 80 meters, and the bubble is located in the middle of the domain, 400 m above the ground. A positive buoyancy of the bubble is imposed via the potential temperature excess. It reaches 3 K in the center of the bubble and decreases to 0 K at its edge in a sine-like manner. Ambient relative humidity is constant and equal to 92%. Under such conditions, a relatively fast cloud formation due to saturation adjustment is obtained (see Fig. 2). A process of rain formation is based on Kessler parametrization (Grabowski and Smolarkiewicz [6]).

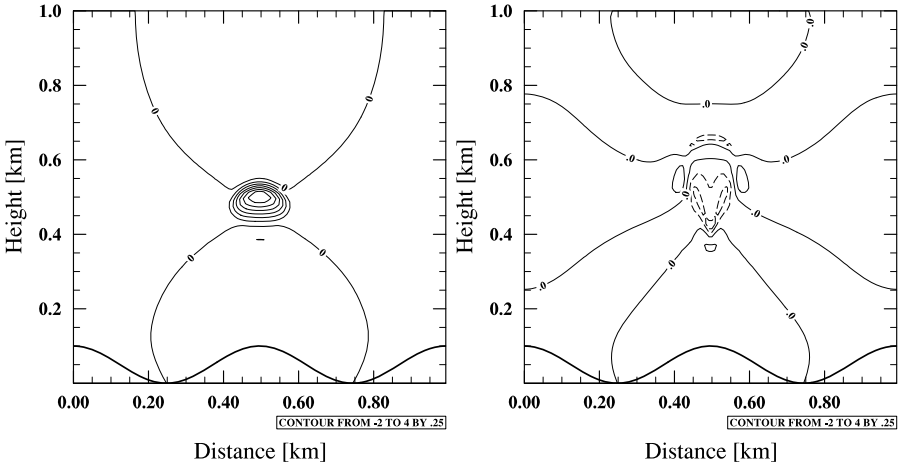


Fig. 1. The distribution of potential temperature perturbation (in Kelvins) after 30 s (left) and 150 s (right) of thermal evolution. XZ crossection at Y=496 m is shown. Dash pattern marks negative perturbation values. The orography is shown with a solid, bold line.

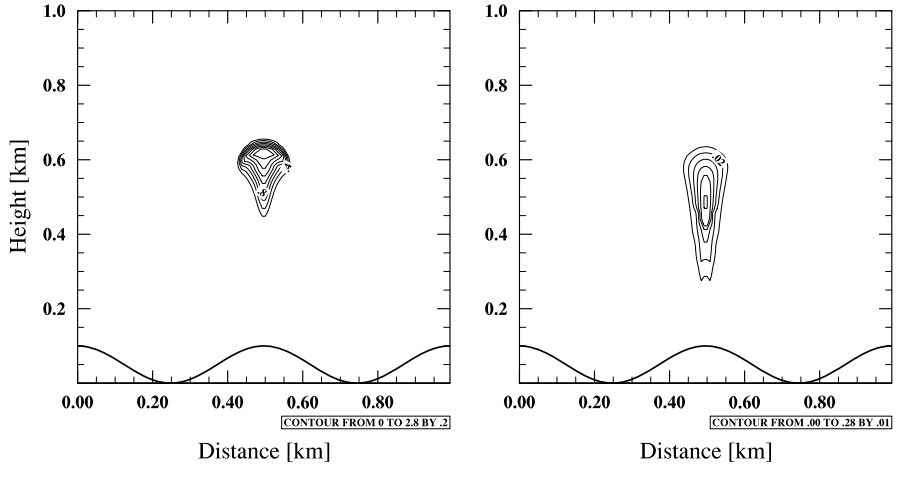


Fig. 2. Cloud (left) and rain (right) water mixing ratio in $[\frac{g}{kg}]$ after 150 s of thermal development. Due to the saturation adjustment, a cloud condensation occurs and precipitation starts to fall from the cloud.

The parallelization of the code is based on two-dimensional horizontal decomposition of the computational grid, where every subdomain spans the whole vertical extent of the domain and a part of its horizontal area. This is a typical approach in NWP modelling (EULAG version implementing three dimensional domain decomposition was developed recently by Piotrowski et al. [7]). Each subdomain has the same grid number. It is extended horizontally with a halo (*i.e.* ghost cells) and the data involved are exchanged between the neighboring processes. The communication is provided by the Message Passing Interface (MPI) protocol.

The study documents three types of experiments which consider (i) parallel performance of the basic EULAG numerical procedures, (ii) the influence of single precision computations and (iii) the influence of subdomain's shape on the numerical performance of the code.

2.2 Tools and Resources

The numerical model was tested on Cray XT4 supercomputer at the Swiss National Supercomputing Center. Its architecture is based on an AMD Opteron CPU running at a frequency of 2.3 GHz. Opteron nodes may access 8GB of RAM. The CSCS XT4 system consists of 160 quad-core nodes giving 640 processing elements. By default all cores per node were used. The tests employing only one core per node are described in section 3.3. The system is managed by Cray Linux operating system. Each process is executed on a separate, dedicated core.

The executable code was generated using the Portland Group Inc. Fortran 90/95 compiler and linked with the available MPICH implementation of MPI communication protocol. The source code was compiled and linked using by default the third level of optimization, double precision computations and loop vectorization.

The results of the experiments were collected using the CrayPat software. The timing results are exclusive, meaning that the timing of a subroutine does not include timings of called subroutines. The results concerning MPI denote the time spent on executing the MPI library functions. The measurements discussed in section 3.3 were collected with the Integrated Performance Monitoring software.

2.3 Tested Subroutines

A set of the most complex and numerically expensive EULAG subroutines was chosen for testing. It includes the procedures of dynamical core, that is an iterative elliptic pressure solver employing generalized conjugate residual method (GCRK; Smolarkiewicz et al. [8]), advection subroutine (MPDATA3) employing multidimensional positive defined transport algorithm (Smolarkiewicz et al. [9]). The preconditioner (PRECON_BCZ) and Laplace equation solver (LAPLC) are the components of pressure solver (GCRK). Also the subroutine handling phase changes (PRECIP) was tested.

3 Results

3.1 Parallel Performance

First, we examined the scalability of EULAG procedures for different number of processes ranging from 4 up to 512 while the grid number is constant and equals 256×256×64 grid points. Thus, the total number of grid points is about 4.19×10^6. Although the simulated problem is idealized, its numerical complexity is comparable with a typical problem of a regional numerical weather forecast.

Figure 3 (left) presents CPU average user time as a function of number of processes, for different subroutines. For MPI the y-axis denotes the total MPI communication cost. The figure shows that all computational subroutines scale very close to the ideal scaling, which can be evaluated with an efficiency metric (see [10]), adapted to our study:

$$E_P = \frac{4 \cdot T_4}{P \cdot T_P}$$

where P denotes the number of processes used and T_P denotes average CPU user time for P processes in the simulation. In the ideal case E_P is 1. The averaged efficiency E_P, calculated from the Figure 3 is: 0.95 for GCRK, 0.91 for MPDATA3, 0.89 for LAPLC, 1.01 for PRECON and 0.98 for PRECIP.

However, the MPI costs decrease significantly slower comparing with the computational procedures. This results from the increase in the ratio of halo size to subdomain's grid size and shows that one should keep this ratio reasonably small. Otherwise weaker MPI scalability becomes a bottleneck that decreases overall parellel performance of the code and finally saturates it.

The MPI costs are investigated in more detail via analysis of the scalability of some of its components (Fig. 3, right). Some of the procedures like MPI_SEND and MPI_WAIT scale similarly to the overall MPI cost. There are, however, components as, *e.g.* the MPI_COMM_SPLIT subroutine, which scale even weaker, as its cost generally grow with the process number.

Additional series of experiments concerned a performance of the EULAG code for different grid numbers, varying from 32×32×64 to 1024×512×64 grid points. Figure 4 shows the scalability of the CPU average user time for the GCRK subroutine, for selected number of processes. The solid lines and the labels correspond to tests employing the same grid- but different process numbers, all using four cores per node. The basic conclusion is that for a wide scope of grid sizes, used, the GCRK subroutine scales close to the ideal scaling. The dashed lines in Fig. 4 correspond to additional tests employing one core per node and performed with the computational domains of 64×32×64, 256×256×64 and 1024×512×64 grid points. The performance comparison is based on pairs of lines, a solid one and a dashed one, that refer to the identical computational domain. There are three such pairs and in each case the CPU average time of the experiment using four cores per node is approximately two times longer than of the experiment using one core per node. This fact suggests the presence of memory bus saturation when four cores per node are used. It is a situation when the node's

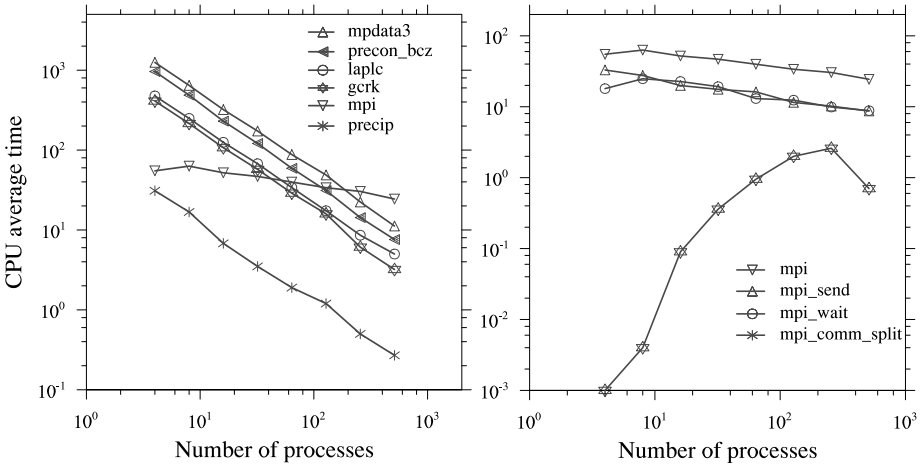


Fig. 3. The scalability of selected EULAG subroutines in terms of the average execution time (left). The scalability of selected MPI library subroutines in the context of EULAG scalability (right).

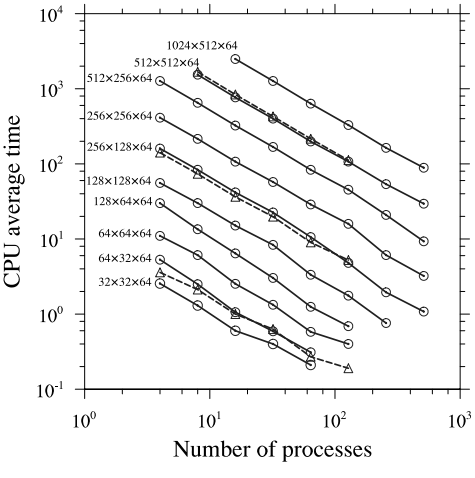


Fig. 4. GCRK subroutine scalability for different domain sizes and for different number of processes ranging from 4 to 512. The labels describe the grid size and concern only tests marked with solid lines. For the description of dashed lines see section 3.1.

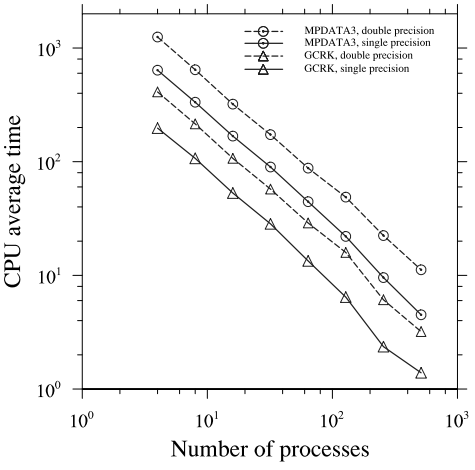


Fig. 5. The comparison of EULAG performance in single and double precision

memory bus cannot handle on time data request from its cores and resulting delays generally affect the GCRK performance. When only one core per node is employed the user time of simulation decreases due to lower number of data requests served by a node. The problem of memory bus saturation is generally regarded as an important one in NWP context and is studied *e.g.* in the frame of the current operational COSMO model development (Oliver Fuhrer, personal communication).

3.2 EULAG Efficiency in Single and Double Precision

The results discussed above were obtained using double precision (DP) floating point representation. Such degree of accuracy is commonly used for the NWP simulations. In order to evaluate the EULAG scalability in single precision (SP), a specific experiment was performed. One can expect that the numerical model in single precision is capable to work about two times faster than in double precision. That results from several reasons. First, the amount of data to be sent and received via MPI is approximately two times smaller whereas any constant communication coefficient does not change. Second, the SSE (see [11]) registers (used in loop vectorization) in SP perform two times more floating point operations per processor's cycle than in DP. Third, the memory bus bandwidth remains the same whereas the byte size of floating point number decreases twice. From the practical view point, however, the degree of precision may have an impact on the number of iterations needed by iterative subroutines such as GCRK.

Figure 5 presents parallel performance of the EULAG code both in single and double precisions. We examine CPU user average time for two numerically expensive subroutines: GCRK and MPDATA3 (the timings do not include PRECON_BCZ, LAPLC and MPI execution time). It appears that single-precision calculations are faster by an average factor 2.1 for MPDATA3 and 2.2 for GCRK.

Table 1. The influence of subdomain shape on the average user time. The results are normalized by reference values for the rectangular domain.

	8x8	4x16	16x4
GCRK	100%	96%	109%
MPDATA3	100%	99%	105%
PRECIP	100%	90%	102%
PRECON_BCZ	100%	97%	107%
LAPLC	100%	96%	109%
EULAG	100%	98%	111%

These results are better than suggested by the theoretical consideration above. That may derive from cache performance as the number of data cache misses (Sec. 3.3) decrease by a factor bigger than 2. Indeed in SP the cache is capable to store twice more floating numbers than in DP, so the data exchange between RAM memory and the caches of different levels is less frequent and the probability of a data miss event is smaller.

3.3 Influence of the Shape of Single Core Subdomains

Finally, the influence of a shape of subdomains served by a single core on EULAG parallel performance is analyzed. The experiment was performed using a computational domain of 512×512×64 grid points and fixed number of processes. The total number of 64 processes is arranged in 3 types of horizontal arrays: 8×8, 4×16 and 16×4.

Table 1 presents the results: user times for EULAG as a whole and for its main subroutines, normalized by the values from the case employing the squared-shaped subdomain (*i.e.* 8×8 computational grid). The table shows that subdomains elongated in the x-direction increase slightly the computational efficiency of the overall code by about 2%. It concerns also the subroutines of the dynamical core, while the efficiency increase of PRECIP is 10%. Subdomains elongated in the y-direction deteriorate the computational efficiency of the overall code by about 11%, which is a significant value. It concerns also the subroutines of the dynamical core. That suggests that the operational optimization of the EULAG code may benefit from an optimization of the shape of the computational subdomains served by a single core.

To explain this behavior, a study of relation between code timing and hardware events was performed. Two expensive loops over a computational domain were analysed: one loop from GCRK and one from PRECON_BCZ. Their timings were compared with the number of hardware events PAPI_L1_DCM (a hardware signal that the processor failed to read or write a piece of data in the L1 cache and was forced to access the piece of data with much longer latency) and PAPI_L2_DCM [12]. The results are presented in Fig. 6. There is a strong correlation between the execution time and the number of both hardware events, which shows that the array operations in Fortran are non-symmetric *i.e.* adjacent data

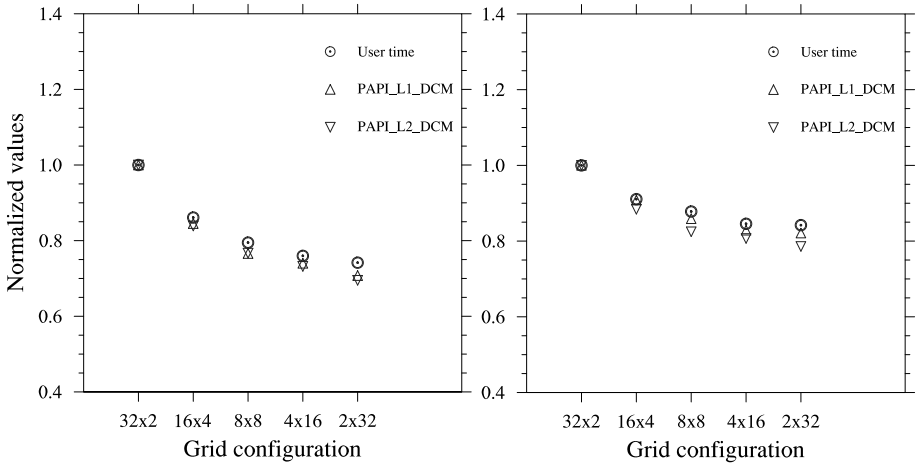


Fig. 6. The normalized user time, the normalized PAPI_L1_DCM and PAPI_L2_DCM number of events for selected loop in the subroutine GCRK (left) and in the subroutine PRECON_BCZ (right). The x-axis consists of 5 tested grid configurations. The total number of processes is 64.

in memory correspond to adjacent data in the x-direction. That explains higher efficiency for subdomains elongated in x-direction.

4 Summary

This paper documents scalability of the EULAG model, a prospective dynamical core of a future COSMO operational weather forecast model, for an idealized weather–like benchmark experiment. The main procedures of the dynamical core and an example physical parameterization of moist processes were examined. The scalability of the computational procedures turned out to be close to the ideal one, while the scalability of the MPI procedures is significantly weaker. In consequence, the overall scalability of the numerical model depends on the ratio of the halo size to the single core subdomain size. In EULAG simulations this ratio should be kept possibly small, so that the MPI cost is small compared to the cost of the dynamical kernel. Next, the memory bus saturation problem is detected and its influence is quantified.

Tests for single precision floating point representation showed an increase of EULAG performance by a factor 2.1 and 2.2 for the main dynamical core subroutines MPDATA3 and GCRK, respectively, comparing with double precision calculations.

Finally, it was shown that the shape of computational subdomains served by single core influences parallel performance of the EULAG, due to the asymmetry of Fortran array operations. Optimization of that shape can benefit general optimization of the model parallel performance.

Acknowledgments. A helpful discussion with dr Zbigniew Piotrowski is gratefully acknowledged. The study was supported by a computational grant from the Swiss National Supercomputing Center-CSCS under s83 project.

References

1. Prusa, J.M., Smolarkiewicz, P.K., Wyszogrodzki, A.A.: EULAG, a computational model for multiscale flows. Comp. Fluids 37(9), 1193–1207 (2008)
2. The Consortium for Small-scale Modeling (September 2011), http://www.cosmo-model.org
3. Piotrowski, Z.P., Kurowski, M.J., Rosa, B., Ziemianski, M.Z.: EULAG Model for Multiscale Flows – Towards the Petascale Generation of Mesoscale Numerical Weather Prediction. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009. LNCS, vol. 6068, pp. 380–387. Springer, Heidelberg (2010)
4. Held, I.M., Suarez, M.J.: A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models. Bull. Amer. Meteor. Soc. 75(10), 1825–1830 (1994)
5. Gal-Chen, T., Somerville, R.C.J.: On the use of a coordinate transformation for the solution of the navier-stokes equations. Journal of Computational Physics 17(2), 209–228 (1975)
6. Grabowski, W.W., Smolarkiewicz, P.K.: A multiscale anelastic model for meteorological research. Mon. Weather Rev. 130, 939–956 (2002)
7. Piotrowski, Z.P., Wyszogrodzki, A.A., Smolarkiewicz, P.K.: Towards petascale simulation of atmospheric circulations with soundproof equations. Acta Geophys. 59(6), 1294–1311 (2011)
8. Smolarkiewicz, P.K., Skamarock, W.C., Klemp, J.B.: Preconditioned Conjugate—Residual Solvers for Helmholz Equations in Nonhydrostatic Models. Mon. Weather Rev., 587–599 (1996)
9. Smolarkiewicz, P.K., Margolin, L.G.: MPDATA: A finite-difference solver for geophysical flows. J. Comput. Phys. 140, 459–480 (1998)
10. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing, 2nd edn. Addison-Wesley (2003)
11. INTEL 64 and IA–32 Architectures Optimization Reference Manual (2009)
12. Performance Application Programming Interface (September 2011), http://icl.cs.utk.edu/papi

Parallel Execution in Metaheuristics for the Problem of Solving Parametric Interval Linear Systems

Jerzy Duda and Iwona Skalna

AGH University of Science and Technology, Krakow, Poland
jduda@zarz.agh.edu.pl,
skalna@agh.edu.pl

Abstract. This paper addresses the problem of solving linear algebraic systems whose elements are nonlinear functions of parameters varying within prescribed intervals. Evolutionary algorithm, differential evolution and two variants of simulated annealing are applied to approximate the hull solution of such systems and compared in terms of accuracy and efficiency. As the computation time for larger problems is significant, calculations for optimisation problems in the family of equations are done in parallel. Structural engineering case studies and numerical experiments involving large uncertainties are carried out to demonstrate accuracy of the metaheuristics and the impact of the parallelization.

1 Introduction

Imprecision, approximation, or uncertainties in the knowledge of the exact values of physical and technical parameters can be modelled conveniently by intervals. In interval analysis, an unknown or imprecise parameter $\tilde{p}$ is replaced by an interval number $\boldsymbol{p} = [\underline{p}, \overline{p}]$ which bounds it. Then, the radius $r(\boldsymbol{p}) = (\overline{p} - \underline{p})/2$ is a measure for the absolute accuracy of the midpoint $\check{p} = (\overline{p} + \underline{p})/2$ considered as an approximation of an unknown value $\tilde{p}$ contained in $\boldsymbol{p}$. Intervals are also commonly used to keep track of and handle rounding errors directly during the calculation.

Now, consider parametric linear systems

$$A(p)x(p) = b(p), \tag{1}$$

where $A(p)$ is an $n \times n$ matrix, $b(p)$ is n-dimensional vector, and $a_{ij}(p)$, $b_i(p)$ are nonlinear functions of $p = (p_1, \ldots, p_k)^T$ which is a k-dimensional vector of parameters. Solving such systems is an important part of many scientific and engineering problems. If some of the parameters are assumed to be unknown but bounded, $p_i \in \boldsymbol{p}_i$ $(i = 1, \ldots, k)$, the following family of parametric linear system, usually called *parametric interval linear system (PILS)*, is obtained

$$A(p)x(p) = b(p),\ p \in \boldsymbol{p}. \tag{2}$$

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 429–438, 2012.

The set of all solutions to the point linear systems from the family (2) is called a *parametric solution set* and is defined as

$$S_{\boldsymbol{p}} = \{x \in \mathbb{R}^n \mid \exists\, p \in \boldsymbol{p} \;\; A(p)x = b(p)\}\,. \tag{3}$$

This set is generally of a complicated nonconvex structure [1]. In practise, therefore, an interval vector $\boldsymbol{x}^*$, called the *outer solution*, satisfying $\square S_{\boldsymbol{p}} \subseteq \boldsymbol{x}^*$ is computed. It is closed and bounded if $A(p)$ is nonsingular for each $p \in \boldsymbol{p}$. An interval vector $\boldsymbol{x}^{**}$, called the *inner solution*, satisfying $\boldsymbol{x}^{**} \subseteq \square S_{\boldsymbol{p}}$ can be calculated as well [12]. If inner and outer solutions are close, this shows good quality of both of them. If, however, the outer solution is much wider than the inner one, one or both of them are poor.

The tightest, w.r.t. the inclusion property, outer solution is called a *hull solution* (or simply a hull) and is denoted by $\square S_{\boldsymbol{p}} = \bigcap \{\boldsymbol{Y} \mid S_{\boldsymbol{p}} \subseteq \boldsymbol{Y}\}$. Computing the hull solution is in general case NP-hard [11]. Some algorithms for its computation were suggested in the literature, e.g., the combinatorial approach, the monotonicity method of Rao & Berke [10], and the vertex sensitivity method of Pownuk [8]. Although, they are often exact at small uncertainty, they do not always give the hull and may underestimate, even at small uncertainties [6].

The problem of computing the hull solution can be written as a problem of solving $2n$ constrained optimisation problems. The following theorem holds true.

Theorem 1. *Let $A(p)x(p) = b(p)$, $p \in \boldsymbol{p}$, be a family of parametric linear systems, and let*

$$\underline{x}_i = \min\{x_i(p) \mid x(p) \in S_{\boldsymbol{p}}\}\,, \tag{4a}$$

$$\overline{x}_i = \max\{x_i(p) \mid x(p) \in S_{\boldsymbol{p}}\}\,. \tag{4b}$$

Then $\square S_{\boldsymbol{p}} = [\underline{x}, \overline{x}]$. □

Optimisation problems (4a) and (4b) are solved in this study using metaheuristic strategies described in Section 2. Some of them (evolutionary optimisation, simulated annealing and tabu search) have been already compared in [13] by the authors. Here, more advanced metaheuristics are applied. They are competitive with the EO method that performed the best in the previous comparison. Section 3 presents structural engineering case studies and numerical experiments. The accuracy of the results and the convergence speed of the metaheuristics are analysed to give hints about their efficiency when solving parametric linear systems with large parameter uncertainties. The paper ends with concluding remarks.

2 Parallel Metaheuristic Strategies

Fitness evaluation relies on solving parametric linear system. Firstly, to save some CPU time, the modified version of Gaussian Elimination (GE) is used here. The back-substitution step starts from the last equation and solves $n - i + 1$ equations, where i is the number of the optimisation problem (4a) or (4b). Secondly, calculations for each of the equations can be done in parallel since they are independent of each other.

2.1 Evolutionary Optimisation

Population P consists of pop_{size} individuals characterised by k-dimensional vectors of parameters $p_i = (p_{i1}, \ldots, p_{ik})^T$, where $p_{ij} \in \boldsymbol{p}_j$, $i = 1, \ldots, pop_{size}$, $j = 1, \ldots, k$. Elements of the initial population are generated at random based on the uniform distribution. The 10% of the best individuals pass to the next generation and the rest of the population is created by the *non-uniform mutation* and *arithmetic crossover* [13]. It turned out from numerical experiments that mutation rate r_{mut} should be close to 1 (r_{mut}=0.95), and the crossover rate r_{crs} should be less than 0.3 (r_{crs}=0.25). Population size and the number of generations depend strongly on the problem size. General outline of the algorithm is shown in Fig. 1.

```
Initialise P of pop_size at random
while (i < n) do
   Select P' from P; Choose parents p1 and p2 from P'
   if (r[0,1] < r_crs) then Offspring o1 and o2 ⟵ Recombine p1 and p2
   if (r[0,1] < r_mut) then Mutate o1 and o2
end while
```

Fig. 1. Outline of an evolutionary algorithm

2.2 Differential Evolution

Since differential evolution (DE) was found to be a very effective optimisation method, especially for continuous problems [3], it was interesting to compare it with EO. DE itself can be treated as a variation of evolutionary algorithm, as the method is founded on the principles of selection, crossover, and mutation. However, in DE the main process focuses on the way new individuals are created. Several strategies were defined for this operation [9]. Basic strategy [3], described as */rand/1/bin* (/a vector for a trial solution is selected in a random way/one pair of other vectors is taken for mutation/binomial crossover is used) creates a mutated individual p_m as follows

$$p_m = p_1 + s(p_2 - p_3) \ , \tag{5}$$

where s is a scale parameter called also an "amplification factor". After a series of experiments, the best strategy for the problem of solving PILS appeared to be a strategy described as */rand-to-best/1/bin*. In this strategy a mutated individual is created on the basis of the best solution found so far and three other, randomly chosen individuals

$$p_m = s(p_{best} - p_1) + s(p_2 - p_3) \ . \tag{6}$$

The mutated individual p_m is then mixed with the original individual p with a probability C_R using the following *binomial crossover*

$$p'_j = \begin{cases} p_{mj}, & \text{if } r \geqslant C_R \text{ or } j = r_n \\ p_j, & \text{if } r > C_R \text{ and } j \neq r_n \end{cases} \ , \tag{7}$$

where $r \in [0, 1]$ is a random number and $r_n \in (0, 1, 2, ..., D)$ is a random index ensuring that p'_j is a at least an element obtained by p_{mj}. After a series of experiments, the following parameters values were taken s=0.8 and C_R=0.9.

```
Initialise P of pop_size at random
while (i < n) do
  while (p_m is not valid) do
    Choose 3 individuals at random p_1, p_2, p_3
    Generate mutant p_m from p and from p_1, p_2, p_3
  end while
  p' ⟵ Crossover(p, p_m)
  if (f(p') > f(p_i)) then p_{i+1} ⟵ p' else p_{i+1} ⟵ p_i
end while
```

Fig. 2. Outline of a differential evolution algorithm

2.3 Simulated Annealing

Preliminary results given by a standard simulated annealing (SA) algorithm ([3], [5]) proved to be poor. Therefore, the authors developed a modified SA algorithm (Fig. 3). It starts with p chosen as the best out of pop_{size} solutions instead of a random one. Additionally, inner $(i = 0, \ldots, n)$ and outer $(j = 0, \ldots, m)$ iterations are used. After each outer iteration the current solution p is reset to, p_{best}, the best solution found so far. The perturbed solution p' is obtained by altering a randomly chosen element p_j of $p = (p_1, \ldots, p_j, \ldots, p_k)^T$. The following parameters values were taken for all experiments: pop_{size}=20, initial temperature t_0=0.9, and degradation ratio d_r= 0.995.

```
Initialise P of pop_size at random;   t ⟵ t_0
Find best solution p from P; p_best ⟵ p
while (i < n) do
  while (j < m) do
    p' ⟵ Perturbate(p)
    if (f(p') > f(p_best)) then p ⟵ p'  f(p_best) = f(p'); p_best ⟵ p;
    else if (r < exp(|f(p') − f(p_best)|/t) then p ⟵ p'; Decrease(t); t ⟵ td_r
  end while
  p ⟵ p_best
end while
```

Fig. 3. Outline of a simulated annealing algorithm

2.4 Adaptive Simulated Annealing

Despite various modifications applied to the standard SA algorithm, the obtained results were much worse than those given by the evolution-based methods. Thus, a more efficient version of the standard SA algorithm, Adaptive Simulated Algorithm (ASA) in the version developed by Ingber ([4]) (see Fig. 4), was next used.

The main idea behind self adaptive algorithms is to automatically adjust the parameters that control the annealing process of the SA algorithm. In ASA it is done by the re-annealing process, which systematically rescales both annealing time and annealing temperature. A new vector is created by altering an old solution as follows

$$p'_j = p_j + y_j \left(\overline{p}_j - \underline{p}_j \right) \quad , \tag{8}$$

where $y_j = \mathrm{sgn}(r - 0.5) t_j \left((1 + \frac{1}{t_j})^{|2r-1|} - 1 \right)$ and $r \in [0,\, 1]$ is a random number. The main parameter influencing the computation time in ASA is the limit of generated states (L_{gen}) and the maximum number of states accepted before quitting (L_{acc}).

```
Initialise p at random;   t = t_0
while (Stop condition is not met) do
  while (i < L_gen) do
    j = 0
    p' ⟵ CreateNew(p)
    Accept or reject p' according to acceptance function
    if (j >= L_acc) then Reannealing
  end while
  Decrease(t)
end while
```

Fig. 4. Outline of an adaptive simulated annealing

3 Numerical Experiments and Results

The performance of the methods described in Section 2 is illustrated by numerical solutions of three practical problems from structural engineering. For each problem hull solution was achieved with global optimisation method [14] and then solutions given by evolutionary optimisation (EO), differential evolution (DE), simulated annealing (SA), and adaptive simulated annealing (ASA) were compared. For the sake of the comparison a standard measure of overestimation was used. For two intervals, such that $[a] \subseteq [b]$, the measure is defined as [7]:

$$O_\omega([a], [b]) := 100\%(1 - \omega([a])/\omega([b])) \quad , \tag{9}$$

where $\omega([x])$ gives the length of interval $[x]$. Since stochastic algorithms are used, thus every algorithm performed 30 independent runs for each problem instance and then average overestimation and standard deviation are provided.

3.1 Example 1 - One Bay Portal Frame

As a first example, the small but realistic problem coming from structural engineering is considered. A one bay portal frame with partially constrained connections, shown in Figure 5, was initially analysed by Corliss et al. [2]. In their work,

the authors have assembled a system of linear equations $K \cdot x = F$ corresponding to the portal frame. The global stiffness matrix K is a symmetric matrix with the following elements (only upper triangular elements different from zero are specified):

$$\begin{aligned}
&a_{11} = a_{66} = \tfrac{A_b E_b}{L_b} + \tfrac{12 E_c I_c}{L_c^3} \\
&a_{16} = -\tfrac{A_b E_b}{L_b},\ a_{22} = a_{77} = \tfrac{A_c E_c}{L_c} + \tfrac{12 E_b I_b}{L_b^3} \\
&a_{13} = a_{68} = \tfrac{6 E_c I_c}{L_c^2},\ a_{27} = -\tfrac{12 E_b I_b}{L_b^3} \\
&a_{44} = \alpha + \tfrac{4 E_b I_b}{L_b},\ a_{45} = \tfrac{2 E_b I_b}{L_b} \\
&a_{24} = a_{25} = -a_{47} = -a_{57} = -a_{78} = \tfrac{6 E_b I_b}{L_b^2} \\
&a_{33} = a_{55} = a_{88} = \alpha + \tfrac{4 E_c I_c}{L_c},\ a_{34} = a_{58} = -\alpha
\end{aligned} \tag{10}$$

The elements of K are rational functions of the following parameters: material properties E_b, E_c, cross-sectional properties A_b, A_c, I_b, I_c, lengths L_b, L_c, and joint stiffness α. The right-hand vector $F = (H, 0, 0, 0, 0, 0, 0, 0)^T$ depends only on the applied lateral load H.

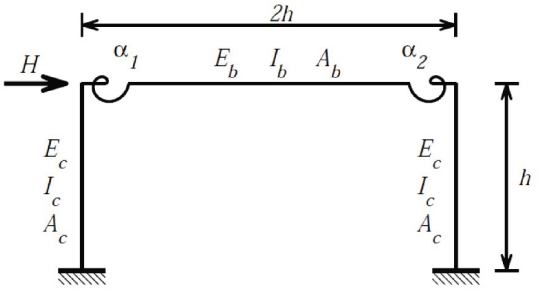


Fig. 5. Simple one bay portal frame with partially constrained connections [2]

Nominal values of the parameters have been taken, as proposed in [2], and uncertainties are given in Table 1. Substituting L_b and L_c with their nominal values yields the problem of solving a parametric linear system

$$K(p)x(p) = F(p), \tag{11}$$

where the parameter $p = (E_b, E_c, A_b, A_c, I_b, I_c, \alpha, H)$.

Comparison of the results of four metaheuristic strategies presented in Section 2 expressed as an average overestimation over the hull solution are given in Table 2. Parameters for particular metaheuristics and computation time are presented as well. All algorithms were coded in C++ using Microsoft Visual Studio 2010 and ran on Intel Xenon X5570 processor with 2.93GHz clock. For the differential evolution method two variants were calculated. In the first (DE1) simple strategy */rand/1/bin/* was used, while in the second (DE2) strategy */rand-to-best/1/bin* was applied.

Table 1. Parameters involved in the portal frame, their nominal values and uncertainties

Parameter		Nominal value	Uncertainty
Young modulus	E_b, E_c	$2.9 \cdot 10^7$ lb/in^2	(40%)
Second moment	I_b	510 in^4	(40%)
	I_c	272 in^4	(40%)
Cross-area	A_b	10.3 in^2	(40%)
	A_c	14.4 in^2	(40%)
External force	H	5,305.5 lbs	(100%)
Joint stiffness	α	$2.77461 \cdot 10^8$ lb-in/rad	(40%)
Length	L_b	288 in	
	L_c	144 in	

Table 2. Results for one bay portal steel frame with uncertainties given in Table 1

Method	Parameters values	Average overestimation	Standard deviation	Computation time [s]
EO	$n = 200$, $pop_{size} = 30$	0.00%	0.00%	0.76
DE1	$n = 200$, $pop_{size} = 30$	0.09%	0.13%	0.59
DE2	$n = 200$, $pop_{size} = 30$	0.00%	0.00%	0.62
ASA	$L_{gen} = 1000$, $L_{acc} = 180$	0.00%	0.00%	0.64
SA	$n = 1000$, $m = 180$	0.27%	0.41%	0.56

Three of five algorithms gave the results which were very close to the hull solution. Basic simulated annealing algorithm performed the worst. Differential evolution strategy denoted as */rand-to-best*/1/*bin* occurred to be better than */rand*/1/*bin* strategy.

3.2 Example 2 - Four-Bay Truss

As a second example, a finite element model for a four-bay two-floor truss, shown in Fig. 6, is considered. There are 15 nodes, 38 elements. The truss is subjected to equal downward forces of 20 kN at nodes 2, 3, and 4. It is assumed that all beams have the same Young modulus $E = 2.0 \cdot 10^{11}$ Pa and the same cross-section area $A = 0.005$ m^2. The length of horizontal beams $L = 10$ m, and vertical beams are a half of the horizontal ones. Material properties and loads are assumed to be uncertain by 40% and 60%, respectively, resulting in 41 uncertain parameters. The ranges for horizontal and vertical displacements of nodes 3 and 4 are shown in Table 3.

For much larger example than the previous *Example* 1 differential evolution outperformed other metaheuristics. The results achieved by both variants of DE can be equally compared only with the results achieved by adaptive simulated annealing.

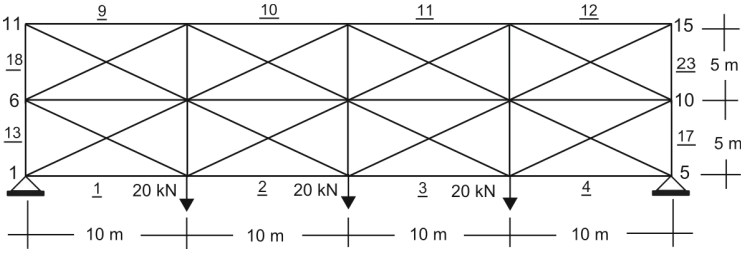


Fig. 6. Four-bay truss two-floor

Table 3. Results for four-bay two-floor truss with 40% uncertainty in material properties and 60% uncertainty in loads

Method	Parameters values	Average overestimation	Standard deviation	Computation time [s]
EO	$n = 30$, $pop_{size} = 16$	18.32%	6.21%	163
DE1	$n = 300$, $pop_{size} = 30$	1.37%	0.57%	153
DE2	$n = 300$, $pop_{size} = 30$	0.01%	0.02%	155
ASA	$L_{gen} = 700$, $L_{acc} = 100$	2.81%	1.04%	156
SA	$n = 15000$, $m = 500$	21.14%	12.74%	135

3.3 Parallelisation

In order to shorten long computation time especially for *Example* 2 in the final experiments the calculations for families of equations were done in parallel. Since all algorithms were coded in C++ using Visual Studio 2010 thus the authors used Parallel Patter Library (PPL) for the parallelisation of the main loop. Computation time for evolutionary optimisation method and simulated annealing utilising only one, two and four processor units (CPU cores) are shown in Table 4.

Table 4. Computation time for different number of cores used for EO and SA

# cores	Example1/EO	Example1/SA	Example2/EO	Example2/SA
1 core	0.76s	0.56s	163s	131s
2 cores	0.42s	0.29s	74s	61s
2/1 ratio	1.81	1.93	2.20	2.15
4 cores	0.23s	0.21s	47s	52s
4/1 ratio	3.30	2.67	3.47	2.52

For evolutionary optimisation method, parallel processing of the main loop on four cores allowed for the shortening of computation time by 3.3–3.5 times compared to single CPU. The same parallelisation method applied for simulated

annealing algorithm was not so effective, allowing for time shortening between 2.5–2.7 times. The results are a little bit surprising as the gain from using the same number of cores was expected to be similar. Such difference can be caused by Parallel Pattern Library algorithm for balancing the tasks, which for small loops like in the presented experiments can have a significant scheduling overhead. Calculations for two cores were done on the same four-cores processor, but the number of concurrent tasks (*MaxConcurrency*) was limited to only two tasks. Evidently the PPL algorithm allowed for using more than two cores for running the two concurrent tasks, so the gain for *Example* 2 was about 2.2 times than for single CPU.

4 Conclusions

The methods based on the theory of evolution (evolutionary algorithm and differential evolution) along with adaptive simulated annealing occurred to be the best heuristic methods for solving parametric interval linear system for relatively small problem (*Example* 1). All those algorithms gave solutions, which were equal or very close to the hull solution. For a much larger problem (*Example* 2) differential evolution and adaptive simulated annealing outperformed evolutionary optimisation method, however, the computation time was significant. If the time restriction was set to a half or smaller than original time, EO algorithm gave similar results. Basic simulated annealing algorithm with some modifications performed very unstable. Sometimes it gave relatively good solutions, but usually the results generated by it could be hardly compared to the others.

Parallelisation of the calculations allowed for a significant shortening of the computation time for both evolutionary method and simulated annealing. Certainly, time saved in this way is limited by the number of equations that are to be calculated and the processor units used. On the other hand, for more processing units further improvement can be also achieved. Simultaneously with the parallelisation of the calculation process for the family of equations, parallelisation can be also applied to the metaheuristics processes themselves, e.g. for the evaluation of the fitness function. Naturally, it makes sense only for the metaheuristics operating on a population of solutions, like evolutionary algorithms or differential evolution, not the ones operating on a single solution like simulated annealing.

References

1. Alefeld, G., Kreinovich, V., Mayer, G.: The Shape of the Solution Set for Systems of Interval Linear Equations with Dependent Coefficients. Mathematische Nachrichten 192(1), 23–36 (2006)
2. Corliss, G., Foley, C., Kearfott, R.B.: Formulation for Reliable Analysis of Structural Frames. Reliable Computing 13(2), 125–147 (2007)
3. Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: Metaheuristics for Hard Optimization. Springer (2006)

4. Ingber, L.: Adaptive simulated annealing (ASA): Lessons learned. J. Control and Cybernetics 25(1), 33–54 (1996)
5. Luke S.: Essentials of Metaheuristics. eBook 226 pages (2009), http://cs.gmu.edu/~sean/book/metaheuristics/
6. Neumaier, A., Pownuk, A.: Linear Systems with Large Uncertainties with Applications to Truss Structures. Reliable Computing 13(2), 149–172 (2007)
7. Popova, E., Iankov, R., Bonev, Z.: 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), pp. 245–265 (2006)
8. Pownuk, A.: Efficient Method of Solution of Large Scale Engineering Problems with Interval Parameters Based on Sensitivity Analysis. In: Muhanna, R.L., Mullen, R.L. (eds.) Proc. NSF Workshop Reliable Engineering Computing, Savannah, GA, September 15-17, pp. 305–316 (2004)
9. Price, K.S., Rainer, M., Lampinen, J.A.: Differential Evolution. A Practical Approach to Global Optimization. Springer, Berlin (2005)
10. Rao, S.S., Berke, L.: Analysis of uncertain structural systems using interval analysis. AIAA Journal 35, 727–735 (1997)
11. 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)
12. Rump, S.M.: Verification Methods for Dense and Sparse Systems of Equations. In: Herzberger, J. (ed.) Topics in Validated Computations, pp. 63–135. Elsevier Science B. V. (1994)
13. Skalna, I., Duda, J.: A Comparison of Metaheurisitics for the Problem of Solving Parametric Interval Linear Systems. In: Dimov, I., Dimova, S., Kolkovska, N. (eds.) NMA 2010. LNCS, vol. 6046, pp. 305–312. Springer, Heidelberg (2011)
14. Skalna, I., Pownuk, A.: Global optimization method for computing interval hull solution for parametric linear systems. International Journal of Reliability and Safety 3, 235–245 (2009)

Organizing Calculations in Algorithms for Solving Systems of Interval Linear Equations Using the "Interval Extended Zero" Method

Ludmila Dymova and Mariusz Pilarek

Institute of Computer & Information Sciences
Czestochowa University of Technology
Dabrowskiego 73, 42-200 Czestochowa, Poland
dymova@icis.pcz.pl

Abstract. In this report, an approach to organization of calculations in algorithms for solving systems of interval linear equations based on the "interval extended zero" is proposed.This approach provides narrowest possible interval results. The "interval extended zero" method was used earlier to solve systems of interval linear equations.It was shown that it considerably reduces the undesirable excess width effect. However, successive research have shown that different interval results may be obtained, if some calculations are organized differently. It is also shown that the interval multiplication operation affects the width of the resulting intervals in a similar extent as interval division.

Keywords: System of interval linear equations, Interval extended zero method, Organizing the calculations.

1 Introduction

The "interval extended zero" method was used to solve systems of interval linear equations in [3,4,5]. It was shown that this method not only allows us to get much narrower interval results than those obtained using classic interval division [7], but also makes it possible to avoid inverted interval solutions. This method can be naturally classified as a formal (algebraic) solution of interval linear systems in the framework of interval extended zero method. Successive research have shown that when using "interval extended zero" method different interval results may be obtained from used algorithm if the interval calculations are organized differently.

In the current report, three algorithms for solving systems of interval linear equations are used as examples - Gaussian elimination algorithm, Gauss-Jordan algorithm and LU algorithm. It is shown that the interval multiplication operation affects the width of the resulting intervals in a similar extent as interval division.

The rest of the paper is set out as follows. In Section 2, we present and analyze the results obtained using classical interval extensions of above mentioned

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 439–446, 2012.

algorithms and extensions based on "interval extended zero" method. It is shown that the use of "interval extended zero" method makes it possible to organize the interval calculations in such a way that they provide narrowest possible interval results. Section 3 presents the analysis of influence of interval multiplication and division based on the "interval extended zero" method on the excess width effect. Section 4 concludes with some remarks.

2 Organizing Calculations in Algorithms for Solving Systems of Interval Linear Equations

A system of linear interval equations can be presented as follows:

$$[\mathbf{A}][\mathbf{x}] = [\mathbf{b}], \tag{1}$$

where $[\mathbf{A}]$ is an interval matrix, $[\mathbf{b}]$ is an interval vector and $[\mathbf{x}]$ is an interval solution vector. There are several algorithms that can be used to solve the system (1). The most popular is the Gaussian elimination algorithm [1] which consists of two stages and in the interval form may be presented as follows:

– forward elimination:

$$[s_{ik}] = \frac{[a_{ik}]}{[a_{kk}]}, \tag{2}$$

$$[a_{ij}] = [a_{ij}] - [s_{ik}] \cdot [a_{kj}], \ [b_i] = [b_i] - [s_{ik}] \cdot [b_k], \tag{3}$$

where $0 \notin [a_{kk}]$, $k = 0, 1, ..., n-1; i, j = k+1, k+2, ..., n$,

– backward substitution:

$$[s] = \sum_{j=i+1}^{j=n} [a_{ij}] \cdot [x_j], \quad [x_i] = \frac{([b_i] - [s])}{[a_{ii}]}, \tag{4}$$

where $0 \notin [a_{ii}]$, $i = 0, 1, ..., n-1$,

where $[a_{ii}]$ are the elements of interval matrix $[\mathbf{A}]$, $[b_i]$ are the elements of interval vector $[\mathbf{b}]$ and $[x_i]$ are the elements of the interval solution vector $[\mathbf{x}]$.

This algorithm was used in [5] to solve the Leontief's input-output model [6] in the interval setting. This model can be defined as follows:

$$(I - [A]')[x] = [b],$$

where I is the identity matrix, $[A]'$ is the interval technical coefficient matrix, $[b]$ is the final interval production vector and $[x]$ is the global interval production vector.

Denoting $I - [A]'$ as $[A]$ the system can be solved using Gaussian elimination algorithm. In [5], all interval operations of division in (2) and (4) were replaced with the so-called modified interval division (MID) based on the "interval extended zero" method and the narrow interval solution of the system was obtained.

There are also two other often used algorithms of solving systems of linear equations:

– Gauss-Jordan algorithm [1]:

$$[s_k] = \frac{[1]}{[a_{kk}]}, \quad [a_{ij}] = [a_{ij}] - [s_k] \cdot [a_{kj}] \cdot [a_{ik}] \tag{5}$$

$$[b_i] = [b_i] - [s_k] \cdot [b_k] \cdot [a_{ik}], \quad [x_k] = [b_k], \tag{6}$$

where $0 \notin [a_{kk}]$, $k = 0, 1, \ldots, n$; $j = k+1, k+2, \ldots, n$; $i = 0, 1, \ldots, n$,

– LU algorithm [1], that consists of two stages:
  – forward elimination, Doolittle algorithm [1]:
    calculating the upper triangular matrix:

$$[a_{ki}] = [a_{ki}] - [a_{kj}] \cdot [a_{ji}], \tag{7}$$

    where $k = 0, 1, \ldots, n$, $i = k+1, k+2, \ldots, n$; $j = 0, 1, \ldots, k$,
    calculating the lower triangular matrix:

$$[s_k] = \frac{[1]}{[a_{kk}]}, \quad [a_{ik}] = [a_{ik}] - [a_{ij}] \cdot [a_{jk}], \quad [a_{ik}] = [a_{ik}] \cdot [s_k], \tag{8}$$

    where $0 \notin [a_{kk}]$, $k = 0, 1, \ldots, n$; $i = k+1, k+2, \ldots, n$; $j = 0, 1, \ldots, k$,
  – backward substitution:
    calculating the upper triangular matrix:

$$[b_k] = [b_k] - [b_i] \cdot [a_{ki}], \tag{9}$$

    where $k = 1, 2, \ldots, n; i = 0, 1, \ldots, k$,
    calculating the vector of results:

$$[x_k] = \frac{[b_k] - [a_{ki}] \cdot [b_i]}{[a_{kk}]}, \tag{10}$$

    where $k = n-1, n-2, \ldots, 0; i = k+1, k+2, \ldots, n$.

These three algorithms have been used to solve the example of 6-sector economic system from [8]:

$$A' = \begin{bmatrix} [0.1389, 0.1396] & [0.0804, 0.0806] & [0.0033, 0.0036] & [0.0001, 0.0001] & [0.0321, 0.0327] & [0.0052, 0.0054] \\ [0.1565, 0.1571] & [0.5043, 0.5047] & [0.5634, 0.5643] & [0.3401, 0.3421] & [0.2405, 0.2411] & [0.2642, 0.2654] \\ [0.0001, 0.0002] & [0.0004, 0.0005] & [0.0067, 0.0069] & [0.0013, 0.0015] & [0.0090, 0.0090] & [0.0145, 0.0149] \\ [0.0110, 0.0113] & [0.0178, 0.0178] & [0.0296, 0.0298] & [0.0140, 0.0146] & [0.1103, 0.1110] & [0.0413, 0.0417] \\ [0.0214, 0.0216] & [0.0749, 0.0750] & [0.0917, 0.0917] & [0.0490, 0.0490] & [0.0400, 0.0402] & [0.0454, 0.0458] \\ [0.0268, 0.0271] & [0.0284, 0.0407] & [0.0142, 0.0144] & [0.0358, 0.0363] & [0.1086, 0.1090] & [0.0981, 0.0987] \end{bmatrix}$$

$$b = \{[5000, 5200][91000, 92000][5200, 5500][1100, 1300][3400, 3600][5900, 6180]\}^T \tag{11}$$

Interval division operations in (5), (8) and (10) were replaced with the modified interval division (MID) based on "interval extension zero" method (see [4,5]). This method allows us to obtain much narrower interval results than those obtained using classic interval division [7]. Six implementations have been developed: Gaussian elimination algorithm that uses classic interval division method in (2) and (4) (denoted as $Gauss_{clas}$), Gaussian elimination algorithm that uses MID operations in (2) and (4) ($Gauss_{mod}$), Gauss-Jordan algorithm that uses classic interval division in (5) (GJ_{clas}), Gauss-Jordan algorithm that uses MID operations in (5) (GJ_{mod}), LU algorithm that uses classic interval

division method in ([8] and [10])) (LU_{clas}) and LU algorithm that uses MID operations in ([8] and [10]) (LU_{mod}).

To estimate the quality of obtained results, we provide the special relative index of uncertainty, RIU. It may serve as the quantitative measure of the excess width effect. It was calculated on resulting vectors as a maximal value from all elements: $RIU = \max((x_m - \underline{x}), (\overline{x} - x_m))/x_m \cdot 100\%$, where $x_m = (\underline{x} + \overline{x})/2$. In the considered example, the value of RIU_{in} calculated for the input matrix is equal to 33.3%, for the input vector the value of RIU_{in} is equal to 8.33%.

Table 1 contains the results obtained from the presented example. $Gauss_{clas}$, GJ_{clas} and LU_{clas} algorithms provide widest interval results. The results obtained using GJ_{mod} algorithm were only slightly narrower and the results obtained from $Gauss_{mod}$ and LU_{mod} algorithms were almost 2 times narrower than those obtained with the use of algorithms based on the classic interval division. It is seen that different algorithms provide different results when the modified interval division (MID) based on the "interval extended zero" method is used.

Table 1. Results obtained for the example of 6-sector economic system

$Gauss_{clas}$	$Gauss_{mod}$	GJ_{clas}	GJ_{mod}	LU_{clas}	LU_{mod}
[27941, 28914]	[28286, 28564]	[27941, 28914]	[27957, 28896]	[27941, 28914]	[28286, 28564]
[226211, 232376]	[228230, 230324]	[226211, 232376]	[226315, 232261]	[226211, 232376]	[228230, 230324]
[5815, 6225]	[5926, 6113]	[5815, 6225]	[5817, 6223]	[5815, 6225]	[5926, 6113]
[9079, 9736]	[9288, 9523]	[9079, 9736]	[9087, 9727]	[9079, 9736]	[9288, 9523]
[23674, 24687]	[23981, 24375]	[23674, 24687]	[23689, 24670]	[23674, 24687]	[23981, 24375]
[17798, 21970]	[18769, 20693]	[17798, 21970]	[17850, 21631]	[17798, 21970]	[18769, 20693]
$RIU = 9.87\%$	$RIU = 4.87\%$	$RIU = 9.87\%$	$RIU = 9.57\%$	$RIU = 9.87\%$	$RIU = 4.87\%$

We can see that in all above algorithms the multiplication by the reversed element from the diagonal ($\frac{[1]}{[a_{kk}]}$) is used. It is known [2] that division operation is 5-10 times slower than the multiplication. Therefore, numerical algorithms usually are developed in such a way that they reduce the number of division operations. We can see that presented above algorithms use the minimal number of divisions. Of course, such approach leads to the development of faster algorithm, but it is not obvious that such algorithms can considerable reduce the excess width effect, especially when using MID. To study this issue, we reorganize the calculations using division operations instead of the multiplications:

– forward elimination of Gaussian elimination algorithm

$$[a_{ij}] = [a_{ij}] - \frac{[a_{ik}] \cdot [a_{kj}]}{[a_{kk}]}, \quad [b_i] = [b_i] - \frac{[a_{ik}] \cdot [b_k]}{[a_{kk}]}, \tag{12}$$

where $0 \notin [a_{kk}]^{(k)}$, $k = 0, 1, ..., n-1$; $i, j = k+1, k+2, ..., n$,

– calculation of the matrix $[\mathbf{A}]$ and vector $[\mathbf{b}]$ in the Gauss-Jordan algorithm

$$[a_{ij}] = [a_{ij}] - \frac{[a_{kj}] \cdot [a_{ik}]}{[a_{kk}]}, \quad [b_i] = [b_i] - \frac{[b_k] \cdot [a_{ik}]}{[a_{kk}]}, \tag{13}$$

where $0 \notin [a_{kk}]$, $k = 0, 1, \ldots, n$; $j = k+1, k+2, \ldots, n$; $i = 0, 1, \ldots, n$,

– calculating the lower triangular matrix in the forward elimination of LU:

$$[a_{ik}] = [a_{ik}] - [a_{ij}] \cdot [a_{jk}], \quad [a_{ik}] = \frac{[a_{ik}]}{[a_{kk}]}, \tag{14}$$

where $0 \notin [a_{kk}]$, $k = 0, 1, \ldots, n$; $i = k+1, k+2, \ldots, n$; $j = 0, 1, \ldots, k$.

In the above algorithms, the multiplications by the reversed diagonal elements of $[\mathbf{A}]$ were replaced with the division by the diagonal elements. The results obtained with the use of modified algorithms are presented in Table 2. We can see that $Gauss_{clas}$, GJ_{clas} and LU_{clas} algorithms provide exactly the same results as previously (see Table 1), but the results obtained by other modified algorithms differ from those presented in Table 1. Both $Gauss_{mod}$ and GJ_{mod} algorithms give much narrower results than those presented in Table 1, while LU_{mod} algorithm produces only slightly wider intervals.

Table 2. Results obtained for the example of 6-sector economic system after reorganizing the calculations in the algorithms

$Gauss_{clas}$	$Gauss_{mod}$	GJ_{clas}	GJ_{mod}	LU_{clas}	LU_{mod}
[27941, 28914]	[28294, 28555]	[27941, 28914]	[28190, 28658]	[27941, 28914]	[28286, 28564]
[226211, 232376]	[228367, 230179]	[226211, 232376]	[227809, 230735]	[226211, 232376]	[228227, 230325]
[5815, 6225]	[5933, 6106]	[5815, 6225]	[5918, 6120]	[5815, 6225]	[5926, 6113]
[9079, 9736]	[9303, 9507]	[9079, 9736]	[9259, 9552]	[9079, 9736]	[9288, 9524]
[23674, 24687]	[24009, 24345]	[23674, 24687]	[23971, 24383]	[23674, 24687]	[23978, 24378]
[17798, 21970]	[19198, 20256]	[17798, 21970]	[18797, 20653]	[17798, 21970]	[18766, 20693]
$RIU = 9.87\%$	$RIU = 2.68\%$	$RIU = 9.87\%$	$RIU = 4.71\%$	$RIU = 9.87\%$	$RIU = 4.88\%$

We can see that when using classic interval division operation it doesn't matter how we organize the calculations and which algorithm we use to solve the systems of interval linear equations - we always get same results. However, if we use the MID operation based on "interval extended zero" method, the results depend on the algorithm and the number of multiplication and division operations.

As the another example, let us consider the randomly generated 500-sector economic system. Interval data have been generated as follows: the sum of the elements in columns of the coefficient matrix couldn't be greater than 1, therefore the elements were generated in the range $0 \div 1/n$, where n is the number of rows in the matrix. Elements from the final production vector b were randomly generated in the range $0 \div 2500000$. The bounds of intervals were obtained as follows: $\underline{a}'_{ij} = a'_{ij} - 0.05 \cdot a'_{ij}$, $\overline{a}'_{ij} = a'_{ij} + 0.05 \cdot a'_{ij}$. The values of RIU_{in} for the input matrix and vector were equal to 5%.

In Table 3, the values of RIU, calculation times and the sum of division and multiplication operations for all algorithms in their original forms, that were implemented with the use of expressions (2) - (10) are presented. Similarly to previous example, the greatest value of RIU was obtained for the $Gauss_{clas}$, GJ_{clas} and LU_{clas} algorithms. GJ_{clas} and GJ_{mod} algorithms were slowest, while the other four algorithms were practically evenly fast. In all algorithms, the numbers of interval division operation are less than 0.003% of all interval operations.

Table 3. Results obtained for the tested algorithms for 500-sector economic system ($RIU_{in} = 5\%$)

	$Gauss_{clas}$	$Gauss_{mod}$	GJ_{clas}	GJ_{mod}	LU_{clas}	LU_{mod}
RIU[%]	14.88	6.827	14.88	14.87	14.88	6.827
time [s]	15.391	15.354	23.681	23.705	15.779	15.757
divisions (div)	999	999	500	500	1000	1000
multiplications (mul)	41916000	41916000	62999750	62999750	41916000	41916000
$div/mul \cdot 100\%$ [%]	0.0024	0.0024	0.0008	0.0008	0.0024	0.0024

Applying the expressions (12) - (14) in the tested algorithms we obtain the results presented in Table 4. We can see that the values of RIU obtained using $Gauss_{mod}$, GJ_{mod} and LU_{mod} algorithms differ from those presented in Table 3. The interval results obtained using $Gauss_{clas}$, GJ_{clas} and LU_{clas} algorithms are the same as in Table 3. It is worth noting that although in the last case we have much more interval divisions ($0.3 - 0.4\%$ of all interval operations), the time of calculations practically is not increased.

Table 4. Results obtained for the modified algorithms for 500-sector economic system ($RIU_{in} = 5\%$)

	$Gauss_{clas}$	$Gauss_{mod}$	GJ_{clas}	GJ_{mod}	LU_{clas}	LU_{mod}
RIU%]	14.88	4.48	14.88	6.41	14.88	6.829
time [s]	15.340	15.456	23.454	23.339	15.239	15.634
divisions (div)	125250	125250	250500	250500	125250	125250
multiplications (mul)	41791250	41791250	62749250	62749250	41791250	41791250
$div/mul \cdot 100\%$ [%]	0.3	0.3	0.4	0.4	0.3	0.3

The obtained results show that when using MID operation based on "interval extended zero" method, it is very important to reorganize calculations to provide as much divisions as possible divisions instead of multiplications. Since in the algorithms of solving the systems of interval linear equations the numbers of division operations are less than 0.5% of all operations, the impact on the speed of the calculations will be marginal.

3 Interval Multiplication Operation as the Source of the Excess Width Effect

Since the "interval extended zero" method reduces the excess width effect which is caused mostly by the classic interval division operation, and taking into account obtained results, we can propose the thesis that the interval multiplication affects the increase of intervals width in the same extent as the classic interval

division does. To prove this thesis we shall use the following example. Consider the following simplest algorithm:

$$[s] = [1] / [\alpha], \quad [y_i] = \sum_{i=0}^{n} [x_i] \cdot [s], \tag{15}$$

We can also implement this algorithm differently:

$$[y_i] = \sum_{i=0}^{n} \frac{[x_i]}{[\alpha]}, \tag{16}$$

In the algorithm (15), there is only one division and n multiplications, while in the algorithm (16) there are n divisions without multiplications.

Table 5 contains the values of RIU and calculation times obtained for several randomly generated interval vectors $[x]$ with $RIU_{in} = 5\%$. The tests have been carried out on AMD Athlon 3000+ 1.8GHz machine. The columns labeled as *mod* contain results obtained for both algorithms implemented with the use of MID based on "interval extended zero" method, whereas the columns labeled as *clas* contain results obtained for algorithms implemented with the use of classic interval division. In both algorithms, the values of RIU for *clas* implementations are exactly the same. They are different however for the *mod* implementations. The algorithm (15) with only one division provides much wider results than the algorithm (16).

Hence, if we use the classic interval division it does not matter for a given algorithm how many multiplication or division operations are in the use (as long as their sum is the same) - the results will remain unchanged. However, if we use the MID operation based on "interval extended zero" method, the results will depend on the number of divisions. The more divisions and less multiplications, the narrower interval results will be obtained. Hence, we can say that interval multiplication operation increases the width of interval results in the same extent as the interval division.

Of course, the algorithm (16) is slower, since there are more slow division operations than in the algorithm (15). Since the MID operation based on "interval extended zero" method is also more complicated than classic interval division,

Table 5. Results obtained from algorithms with classic division and MID operation based "interval extended zero" method

Vector size	*Algorithm (15)*				*Algorithm (16)*			
	mod		*clas*		*mod*		*clas*	
	RIU[%]	*time* [s]	RIU[%]	*time* [s]	RIU[%]	*time* [s]	RIU[%]	*time* [s]
10000	9.95	0.0002	17.41	0.0001	6.98	0.0016	17.41	0.0004
100000	6.17	0.0021	7.46	0.0027	3.56	0.0227	7.46	0.0040
1000000	19.06	0.0239	96.21	0.0254	15.04	0.1831	96.21	0.0427
10000000	17.77	0.2658	66.92	0.2822	13.98	1.9192	66.92	0.4505

the *mod* implementation of the algorithm (16) is about four 4 times slower than the *clas* implementation of this algorithm. In the case of the algorithm (15) there is only one division operation and it practically does not affect the speed of the algorithm.

4 Conclusion

This report is devoted to the organizing calculations in the solution of interval linear systems using the "interval extended zero" method to reduce the excess width effect. Three algorithms for solving the systems of interval linear equations are used as an example. It is shown that we can not only obtain different interval results using these algorithms, but the results will depend on the organization of calculations. It is proved that the interval multiplication operation affects the width of the resulting intervals in the same extent as the classic interval division. In situations, where the speed of the calculations is not the most important factor it is better to use more interval divisions operations instead of the multiplications to obtain much narrower results with the use of the "interval extended zero" method.

References

1. Datta, B.: Numerical Linear Algebra and Applications. Brooks/Cole Pub. Co. (1995)
2. Draper, J., Kwon, T.J., Sondeen, J.: Floating-Point Division and Square Root Implementation using a Taylor-Series Expansion Algorithm. Circuits and Systems, 954–957 (2008)
3. Dymova, L., Pilarek, M., Wyrzykowski, R.: Solving Systems of Interval Linear Equations with Use of Modified Interval Division Procedure. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009. LNCS, vol. 6068, pp. 427–435. Springer, Heidelberg (2010)
4. Sevastjanov, P., Dymova, L.: Fuzzy Solution of Interval Linear Equations. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 1392–1399. Springer, Heidelberg (2008)
5. Dymova, L., Sevastjanov, P.: A new method for solving interval and fuzzy equations: linear case. Information Sciences 17, 925–937 (2009)
6. Leontief, W.: Quantitative input-output relations in the economic system of the United States. Review of Economics and Statistics 18, 100–125 (1936)
7. Moore, R.E.: Interval Arithmetic and Automatic Error Analysis in Digital Computing. PhD thesis, Stanford University (1962)
8. Wu, C.C., Chang, N.B.: Grey input-output analysis and its application for environmental cost allocation. European Journal of Operational Research 145, 175–201 (2003)

An Interval Backward Finite Difference Method for Solving the Diffusion Equation with the Position Dependent Diffusion Coefficient

Malgorzata A. Jankowska

Poznan University of Technology, Institute of Applied Mechanics
Piotrowo 3, 60-965 Poznan, Poland
malgorzata.jankowska@put.poznan.pl

Abstract. The paper deals with the interval backward finite difference method for solving the one-dimensional diffusion equation with the position dependent diffusion coefficient and the boundary conditions of the first type. The interval method considered is based on the conventional backward finite difference method. Moreover, it takes into account a formula of a local truncation error of the method. Such local truncation error of the conventional method is bounded by the appropriate interval values. In most scientific applications we cannot find the endpoints of such intervals exactly and it is of great importance to approximate them in the most accurate way. The paper presents a method of such approximation.

Keywords: one-dimensional diffusion equation, position dependent diffusion coefficient, backward finite difference method, interval methods.

1 Introduction

The initial-boundary value problems for partial differential equations are frequently used to describe some physical processes. Therefore, the problems of solving such equations were analyzed by many authors (see e.g. Manikonda, Berz, Makino [7], Nagatou, Hashimoto, Nakao [9], Watanabe, Yamamoto, Nakao [11]). The interval methods based on some finite difference schemes for solving the heat (or diffusion) equation with the constant diffusion coefficient were proposed by Jankowska and Marciniak in [5], [6], [8]. The paper presents a new interval approach to the solution of the diffusion equation with the diffusion coefficient given as a function of position.

2 One-Dimensional Diffusion Equation and Interval Backward Finite Difference Method

Consider the one-dimensional diffusion equation of the form

$$\frac{\partial u}{\partial t}(x,t) - \frac{\partial}{\partial x}\left(\gamma(x)\frac{\partial u}{\partial x}(x,t)\right) = 0, \quad 0 < x < L,\ t > 0, \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 447–456, 2012.

subject to the initial and boundary conditions

$$u(x,0) = f(x), \quad 0 \le x \le L, \tag{2}$$

$$u(0,t) = \varphi_1(t), \; u(L,t) = \varphi_2(t), \quad t > 0, \tag{3}$$

where $\gamma = \gamma(x)$ is a diffusion coefficient. A function $u = u(x,t)$ represents a concentration distribution and it depends on one spatial variable x and time t. The diffusion equations such as a diffusion coefficient is a function of position, as well as concentration or time, are considered in detail in e.g. [1] and [2].

Now, we set the maximum time $T_{\max}$ and choose two integers n and m. Hence, for the the mesh constants h and k such as $h = L/n$ and $k = T_{\max}/m$, the grid points are (x_i, t_j), where $x_i = ih$ for $i = 0, 1, \ldots, n$ and $t_j = jk$ for $j = 0, 1, \ldots, m$. Furthermore, we make the additional assumption on the function $\gamma = \gamma(x)$, i.e. $\gamma \in C^3[0, L]$.

If we introduce the notation $\lambda = k/h^2$, then the exact formula of the backward finite difference scheme for solving (1) with (2)-(3) can be given as follows

$$\begin{aligned} &-\lambda\gamma\left(x_{i-1/2}\right) u(x_{i-1}, t_j) + \left(1 + \lambda\gamma\left(x_{i-1/2}\right) + \lambda\gamma\left(x_{i+1/2}\right)\right) u(x_i, t_j) \\ &\qquad -\lambda\gamma\left(x_{i+1/2}\right) u(x_{i+1}, t_j) = u(x_i, t_{j-1}) + r_{i,j}, \\ &i = 1, 2, \ldots n-1, \; j = 1, 2, \ldots, m, \end{aligned} \tag{4}$$

with the error term of the form

$$\begin{aligned} r_{i,j} = &-\frac{1}{2}k^2 \frac{\partial^2 u}{\partial t^2}(x_i, \eta_j) - \frac{1}{24}h^2 k \frac{d}{dx}\gamma(x_i) \frac{\partial^3 u}{\partial x^3}(\xi_i, t_j) \\ &-\frac{1}{24}hk\left[\frac{d^3}{dx^3}\gamma\left(\zeta_i^{(+)}\right) u\left(\widetilde{\zeta}_i^{(+)}, t_j\right) + 3\frac{d^2}{dx^2}\gamma\left(\zeta_i^{(+)}\right) \frac{\partial u}{\partial x}\left(\widetilde{\zeta}_i^{(+)}, t_j\right)\right. \\ &\quad +3\frac{d}{dx}\gamma\left(\zeta_i^{(+)}\right) \frac{\partial^2 u}{\partial x^2}\left(\widetilde{\zeta}_i^{(+)}, t_j\right) + \gamma\left(\zeta_i^{(+)}\right) \frac{\partial^3 u}{\partial x^3}\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \\ &\quad -\frac{d^3}{dx^3}\gamma\left(\zeta_i^{(-)}\right) u\left(\widetilde{\zeta}_i^{(-)}, t_j\right) - 3\frac{d^2}{dx^2}\gamma\left(\zeta_i^{(-)}\right) \frac{\partial u}{\partial x}\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \\ &\quad \left.-3\frac{d}{dx}\gamma\left(\zeta_i^{(-)}\right) \frac{\partial^2 u}{\partial x^2}\left(\widetilde{\zeta}_i^{(-)}, t_j\right) - \gamma\left(\zeta_i^{(-)}\right) \frac{\partial^3 u}{\partial x^3}\left(\widetilde{\zeta}_i^{(-)}, t_j\right)\right], \end{aligned} \tag{5}$$

where

$$\begin{aligned} &\xi_i \in \left(x_{i-1/2}, x_{i+1/2}\right), \eta_j \in (t_{j-1}, t_j), \\ &\zeta_i^{(+)}, \zeta_i^{(-)} \in \left(x_{i-1/2}, x_{i+1/2}\right), \widetilde{\zeta}_i^{(+)} \in (x_i, x_{i+1}), \widetilde{\zeta}_i^{(-)} \in (x_{i-1}, x_i). \end{aligned} \tag{6}$$

Let us consider the error term (5) with (6) of the backward finite difference scheme. Before we formulate the interval counterpart of (4) we make some additional assumptions on values of the derivatives given in (5) in the midpoints (6).

We assume that for $i = 1, 2, \ldots, n-1$, $j = 1, 2, \ldots m$, there exist the intervals $G_i^{(0)}$, $G_i^{(1)}$, $G_i^{(2)}$, $G_i^{(3)}$, $M_{i,j}^{(+)}$, $M_{i,j}^{(-)}$, $N_{i,j}^{(+)}$, $N_{i,j}^{(-)}$, $P_{i,j}^{(+)}$, $P_{i,j}^{(-)}$, $Q_{i,j}^{(+)}$, $Q_{i,j}^{(-)}$, $Q_{i,j}$, $S_{i,j}$, such as the following relations hold

a) for $\zeta_i^{(+)}, \zeta_i^{(-)} \in \left(x_{i-1/2}, x_{i+1/2}\right)$,

$$\gamma\left(\zeta_i^{(+)}\right) \in G_i^{(0)}, \qquad \gamma\left(\zeta_i^{(-)}\right) \in G_i^{(0)}, \tag{7}$$

$$\frac{d}{dx}\gamma\left(\zeta_i^{(+)}\right) \in G_i^{(1)}, \quad \frac{d}{dx}\gamma\left(\zeta_i^{(-)}\right) \in G_i^{(1)}, \tag{8}$$

$$\frac{d^2}{dx^2}\gamma\left(\zeta_i^{(+)}\right) \in G_i^{(2)}, \quad \frac{d^2}{dx^2}\gamma\left(\zeta_i^{(-)}\right) \in G_i^{(2)}, \tag{9}$$

$$\frac{d^3}{dx^3}\gamma\left(\zeta_i^{(+)}\right) \in G_i^{(3)}, \quad \frac{d^3}{dx^3}\gamma\left(\zeta_i^{(-)}\right) \in G_i^{(3)}, \tag{10}$$

b) for $\widetilde{\zeta}_i^{(+)} \in (x_i, x_{i+1})$, $\widetilde{\zeta}_i^{(-)} \in (x_{i-1}, x_i)$,

$$u\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in M_{i,j}^{(+)}, \quad u\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in M_{i,j}^{(-)}, \tag{11}$$

$$\frac{\partial u}{\partial x}\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in N_{i,j}^{(+)}, \quad \frac{\partial u}{\partial x}\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in N_{i,j}^{(-)}, \tag{12}$$

$$\frac{\partial^2 u}{\partial x^2}\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in P_{i,j}^{(+)}, \quad \frac{\partial^2 u}{\partial x^2}\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in P_{i,j}^{(-)}, \tag{13}$$

$$\frac{\partial^3 u}{\partial x^3}\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in Q_{i,j}^{(+)}, \quad \frac{\partial^3 u}{\partial x^3}\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in Q_{i,j}^{(-)}, \tag{14}$$

c) for $\xi_i \in \left(x_{i-1/2}, x_{i+1/2}\right)$, $\eta_j \in (t_{j-1}, t_j)$,

$$\frac{\partial^3 u}{\partial x^3}(\xi_i, t_j) \in Q_{i,j}, \quad \frac{\partial^2 u}{\partial t^2}(x_i, \eta_j) \in S_{i,j}. \tag{15}$$

Then, the interval backward finite difference method can be given as follows

$$\begin{aligned}
&\left(1 + \lambda\Gamma\left(X_{1/2}\right) + \lambda\Gamma\left(X_{3/2}\right)\right) U_{1,j} - \lambda\Gamma\left(X_{3/2}\right) U_{2,j} = U_{1,j-1} \\
&\qquad + \lambda\Gamma\left(X_{1/2}\right) U_{0,j} + R_{1,j}, \\
&i = 1,\ j = 1, 2, \ldots, m,
\end{aligned} \tag{16}$$

$$\begin{aligned}
&-\lambda\Gamma\left(X_{i-1/2}\right) U_{i-1,j} + \left(1 + \lambda\Gamma\left(X_{i-1/2}\right) + \lambda\Gamma\left(X_{i+1/2}\right)\right) U_{i,j} \\
&\qquad - \lambda\Gamma\left(X_{i+1/2}\right) U_{i+1,j} = U_{i,j-1} + R_{i,j}, \\
&i = 2, 3, \ldots n-2,\ j = 1, 2, \ldots, m,
\end{aligned} \tag{17}$$

$$\begin{aligned}
&-\lambda\Gamma\left(X_{n-3/2}\right) U_{n-2,j} + \left(1 + \lambda\Gamma\left(X_{n-3/2}\right) + \lambda\Gamma\left(X_{n-1/2}\right)\right) U_{n-1,j} \\
&\qquad = U_{n-1,j-1} + \lambda\Gamma\left(X_{n-1/2}\right) U_{n,j} + R_{n-1,j}, \\
&i = n-1,\ j = 1, 2, \ldots, m,
\end{aligned} \tag{18}$$

where

$$\begin{aligned}
R_{i,j} = &-\frac{1}{2}k^2 S_{i,j} - \frac{1}{24}h^2 k D\Gamma(X_i) Q_{i,j} \\
&-\frac{1}{24}hk\left[G_i^{(3)} M_{i,j}^{(+)} + 3G_i^{(2)} N_{i,j}^{(+)} + 3G_i^{(1)} P_{i,j}^{(+)} + G_i^{(0)} Q_{i,j}^{(+)}\right. \\
&\left. - G_i^{(3)} M_{i,j}^{(-)} - 3G_i^{(2)} N_{i,j}^{(-)} - 3G_i^{(1)} P_{i,j}^{(-)} - G_i^{(0)} Q_{i,j}^{(-)}\right],
\end{aligned} \tag{19}$$

and

$$U_{i,0} = F\left(X_i\right),\ i = 0, 1, \ldots, n, \tag{20}$$
$$U_{0,j} = \Phi_1\left(T_j\right),\ U_{n,j} = \Phi_2\left(T_j\right),\ j = 1, 2, \ldots, m.$$

Note that X_i, $i = 0, 1, \ldots, n$, T_j, $j = 0, 1, \ldots, m$ are intervals such that $x_i \in X_i$ and $t_j \in T_j$. Furthermore, $F = F\left(X\right), \Phi_1 = \Phi_1\left(T\right), \Phi_2 = \Phi_2\left(T\right), D\Gamma = D\Gamma\left(X_i\right)$ denote interval extensions of the functions $f = f\left(x\right)$, $\varphi_1 = \varphi_1\left(t\right)$, $\varphi_2 = \varphi_2\left(t\right)$ and $d\gamma/dx\left(x\right)$, respectively.

The interval backward finite difference method can be also given in the following matrix form

$$CU^{(j)} = D^{(j-1)}, \quad j = 1, 2, \ldots, m, \tag{21}$$

where $U^{(j)} = \left[U_{0,j},\ U_{1,j},\ \ldots\ ,\ U_{n,j}\right]^T$,

$$C = \begin{bmatrix} \mu_1 & \nu_{3/2} & 0 & \vdots & 0 & 0 & 0 \\ \nu_{3/2} & \mu_2 & \nu_{5/2} & \vdots & 0 & 0 & 0 \\ 0 & \nu_{5/2} & \mu_3 & \vdots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ddots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \vdots & \mu_{n-3} & \nu_{n-5/2} & 0 \\ 0 & 0 & 0 & \vdots & \nu_{n-5/2} & \mu_{n-2} & \nu_{n-3/2} \\ 0 & 0 & 0 & \vdots & 0 & \nu_{n-3/2} & \mu_{n-1} \end{bmatrix}, \tag{22}$$

$$\mu_i = 1 + \lambda\Gamma\left(X_{i-1/2}\right) + \lambda\Gamma\left(X_{i+1/2}\right),\ \nu_{i\pm1/2} = -\lambda\Gamma\left(X_{i\pm1/2}\right),\ i = 1, 2, \ldots, n-1, \tag{23}$$

and

$$D^{(j-1)} = \begin{bmatrix} U_{1,j-1} + \lambda\Gamma\left(X_{1/2}\right)\Phi_1\left(T_j\right) + R_{1,j} \\ U_{2,j-1} + R_{2,j} \\ U_{3,j-1} + R_{3,j} \\ \cdots \\ U_{n-3,j-1} + R_{n-3,j} \\ U_{n-2,j-1} + R_{n-2,j} \\ U_{n-1,j-1} + \lambda\Gamma\left(X_{n-1/2}\right)\Phi_2\left(T_j\right) + R_{n-1,j} \end{bmatrix}. \tag{24}$$

For the interval backward finite difference method (21) with (22)-(24) we can prove that if the assumptions (7)-(15) about the error term are met, then the exact solution belongs to the interval solution obtained.

3 Error Term Approximation in the Interval Backward Finite Difference Method

Consider the error term (19) of the interval backward finite difference method (16)-(18). In this section we propose a method for the approximation of the endpoints of the intervals given in (19).

We start from the intervals $G_i^{(0)}$, $G_i^{(1)}$, $G_i^{(2)}$ and $G_i^{(3)}$. Since for the function γ given in (1) we assumed that $\gamma \in C^3[0,L]$, then we can introduce the following procedure

$$\begin{aligned}\underline{G}_i^{(0)} &= \min\left\{\gamma(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\\ \overline{G}_i^{(0)} &= \max\left\{\gamma(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\end{aligned} \tag{25}$$

$$\begin{aligned}\underline{G}_i^{(1)} &= \min\left\{d\gamma/dx(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\\ \overline{G}_i^{(1)} &= \max\left\{d\gamma/dx(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\end{aligned} \tag{26}$$

$$\begin{aligned}\underline{G}_i^{(2)} &= \min\left\{d^2\gamma/dx^2(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\\ \overline{G}_i^{(2)} &= \max\left\{d^2\gamma/dx^2(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\end{aligned} \tag{27}$$

$$\begin{aligned}\underline{G}_i^{(3)} &= \min\left\{d^3\gamma/dx^3(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\},\\ \overline{G}_i^{(3)} &= \max\left\{d^3\gamma/dx^3(s) : s \in \left(x_{i-1/2}, x_{i+1/2}\right)\right\}.\end{aligned} \tag{28}$$

Since the function $\gamma = \gamma(x)$ is known from (1), then the derivatives used in (26)-(28) can be derived. Interval values of the interval extensions of the function γ and its derivatives computed at $X_i = \left[x_{i-1/2}, x_{i+1/2}\right]$ can be taken as the intervals $G_i^{(0)}$, $G_i^{(1)}$, $G_i^{(2)}$ and $G_i^{(3)}$, respectively.

Next we consider the intervals $M_{i,j}^{(+)}$ and $M_{i,j}^{(-)}$ such as $u\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in M_{i,j}^{(+)}$, $u\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in M_{i,j}^{(-)}$, where $\widetilde{\zeta}_i^{(+)} \in (x_i, x_{i+1})$, $\widetilde{\zeta}_i^{(-)} \in (x_{i-1}, x_i)$. We can choose their endpoints in the following way

$$\begin{aligned}\underline{M}_{i,j}^{(+)} &\approx \min\left\{u(x_i,t_j), u\left(x_{i+1/2},t_j\right), u(x_{i+1},t_j)\right\},\\ \overline{M}_{i,j}^{(+)} &\approx \max\left\{u(x_i,t_j), u\left(x_{i+1/2},t_j\right), u(x_{i+1},t_j)\right\},\end{aligned} \tag{29}$$

$$\begin{aligned}\underline{M}_{i,j}^{(-)} &\approx \min\left\{u(x_{i-1},t_j), u\left(x_{i-1/2},t_j\right), u(x_i,t_j)\right\},\\ \overline{M}_{i,j}^{(-)} &\approx \max\left\{u(x_{i-1},t_j), u\left(x_{i-1/2},t_j\right), u(x_i,t_j)\right\}.\end{aligned} \tag{30}$$

Then, for the intervals $N_{i,j}^{(+)}$ and $N_{i,j}^{(-)}$ such as $\partial u/\partial x\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in N_{i,j}^{(+)}$, $\partial u/\partial x\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in N_{i,j}^{(-)}$, where $\widetilde{\zeta}_i^{(+)} \in (x_i, x_{i+1})$, $\widetilde{\zeta}_i^{(-)} \in (x_{i-1}, x_i)$, we propose to take

$$\underline{N}_{i,j}^{(+)} \approx \min\left\{N_{i,j}^*, N_{i+1/2,j}^*, N_{i+1,j}^*\right\}, \ \overline{N}_{i,j}^{(+)} \approx \max\left\{N_{i,j}^*, N_{i+1/2,j}^*, N_{i+1,j}^*\right\}, \tag{31}$$

$$\underline{N}_{i,j}^{(-)} \approx \min\left\{N_{i-1,j}^{*}, N_{i-1/2,j}^{*}, N_{i,j}^{*}\right\}, \ \overline{N}_{i,j}^{(-)} \approx \max\left\{N_{i-1,j}^{*}, N_{i-1/2,j}^{*}, N_{i,j}^{*}\right\}, \tag{32}$$

where

$$N_{i,j}^{*} = \frac{\partial u}{\partial x}\left(x_i, t_j\right). \tag{33}$$

For the finite difference approximation of $\partial u/\partial x$ we can use the formulas

$$\frac{\partial u}{\partial x}\left(x_i, t_j\right) = \frac{1}{2h}\left(-3u\left(x_i, t_j\right) + 4u\left(x_{i+1}, t_j\right) - u\left(x_{i+2}, t_j\right)\right) + O\left(h^2\right), \tag{34}$$

$$\frac{\partial u}{\partial x}\left(x_i, t_j\right) = \frac{1}{12h}\left(-u\left(x_{i+2}, t_j\right) + 8u\left(x_{i+1}, t_j\right) - 8u\left(x_{i-1}, t_j\right) + u\left(x_{i-2}, t_j\right)\right) + O\left(h^4\right), \tag{35}$$

$$\frac{\partial u}{\partial x}\left(x_i, t_j\right) = \frac{1}{2h}\left(3u\left(x_i, t_j\right) - 4u\left(x_{i-1}, t_j\right) + u\left(x_{i-2}, t_j\right)\right) + O\left(h^2\right). \tag{36}$$

Now, for the intervals $P_{i,j}^{(+)}$ and $P_{i,j}^{(-)}$ such as $\partial^2 u/\partial x^2\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in P_{i,j}^{(+)}$, $\partial^2 u/\partial x^2\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in P_{i,j}^{(-)}$, where $\widetilde{\zeta}_i^{(+)} \in \left(x_i, x_{i+1}\right)$, $\widetilde{\zeta}_i^{(-)} \in \left(x_{i-1}, x_i\right)$, we propose to take

$$\underline{P}_{i,j}^{(+)} \approx \min\left\{P_{i,j}^{*}, P_{i+1/2,j}^{*}, P_{i+1,j}^{*}\right\}, \ \overline{P}_{i,j}^{(+)} \approx \max\left\{P_{i,j}^{*}, P_{i+1/2,j}^{*}, P_{i+1,j}^{*}\right\}, \tag{37}$$

$$\underline{P}_{i,j}^{(-)} \approx \min\left\{P_{i-1,j}^{*}, P_{i-1/2,j}^{*}, P_{i,j}^{*}\right\}, \ \overline{P}_{i,j}^{(-)} \approx \max\left\{P_{i-1,j}^{*}, P_{i-1/2,j}^{*}, P_{i,j}^{*}\right\}, \tag{38}$$

where

$$P_{i,j}^{*} = \frac{\partial^2 u}{\partial x^2}\left(x_i, t_j\right). \tag{39}$$

For the finite difference approximation of $\partial^2 u/\partial x^2$ we can use the formulas

$$\frac{\partial^2 u}{\partial x^2}\left(x_i, t_j\right) = \frac{1}{h^2}\left(2u\left(x_i, t_j\right) - 5u\left(x_{i+1}, t_j\right) + 4u\left(x_{i+2}, t_j\right) - u\left(x_{i+3}, t_j\right)\right) + O\left(h^2\right), \tag{40}$$

$$\frac{\partial^2 u}{\partial x^2}\left(x_i, t_j\right) = \frac{1}{12h^2}\left(-u\left(x_{i+2}, t_j\right) + 16u\left(x_{i+1}, t_j\right) - 30u\left(x_i, t_j\right) + 16u\left(x_{i-1}, t_j\right) - u\left(x_{i-2}, t_j\right)\right) + O\left(h^4\right), \tag{41}$$

$$\frac{\partial^2 u}{\partial x^2}\left(x_i, t_j\right) = \frac{1}{h^2}\left(2u\left(x_i, t_j\right) - 5u\left(x_{i-1}, t_j\right) + 4u\left(x_{i-2}, t_j\right) - u\left(x_{i-3}, t_j\right)\right) + O\left(h^2\right). \tag{42}$$

Next we consider the intervals $Q_{i,j}^{(+)}$, $Q_{i,j}^{(-)}$, $Q_{i,j}$ such as for $\widetilde{\zeta}_i^{(+)} \in \left(x_i, x_{i+1}\right)$, $\widetilde{\zeta}_i^{(-)} \in \left(x_{i-1}, x_i\right)$, $\xi_i \in \left(x_{i-1/2}, x_{i+1/2}\right)$ we assumed that $\partial^3 u/\partial x^3\left(\widetilde{\zeta}_i^{(+)}, t_j\right) \in$

$Q_{i,j}^{(+)}$, $\partial^3 u/\partial x^3\left(\widetilde{\zeta}_i^{(-)}, t_j\right) \in Q_{i,j}^{(-)}$, $\partial^3 u/\partial x^3\left(\xi_i, t_j\right) \in Q_{i,j}$, respectively. We propose to choose their endpoints as follows

$$\underline{Q}_{i,j}^{(+)} \approx \min\left\{Q_{i,j}^*, Q_{i+1/2,j}^*, Q_{i+1,j}^*\right\}, \ \overline{Q}_{i,j}^{(+)} \approx \max\left\{Q_{i,j}^*, Q_{i+1/2,j}^*, Q_{i+1,j}^*\right\}, \tag{43}$$

$$\underline{Q}_{i,j}^{(-)} \approx \min\left\{Q_{i-1,j}^*, Q_{i-1/2,j}^*, Q_{i,j}^*\right\}, \ \overline{Q}_{i,j}^{(-)} \approx \max\left\{Q_{i-1,j}^*, Q_{i-1/2,j}^*, Q_{i,j}^*\right\}, \tag{44}$$

$$\underline{Q}_{i,j} \approx \min\left\{Q_{i-1/2,j}^*, Q_{i,j}^*, Q_{i+1/2,j}^*\right\}, \ \overline{Q}_{i,j} \approx \max\left\{Q_{i-1/2,j}^*, Q_{i,j}^*, Q_{i+1/2,j}^*\right\}, \tag{45}$$

where

$$Q_{i,j}^* = \frac{\partial^3 u}{\partial x^3}\left(x_i, t_j\right). \tag{46}$$

For the finite difference approximation of $\partial^3 u/\partial x^3$ we can use the formulas

$$\begin{aligned}\frac{\partial^3 u}{\partial x^3}\left(x_i, t_j\right) = \frac{1}{2h^3}\left(-5u\left(x_i, t_j\right) + 18u\left(x_{i+1}, t_j\right) - 24u\left(x_{i+2}, t_j\right) + 14u\left(x_{i+3}, t_j\right)\right.\\ \left.-3u\left(x_{i+4}, t_j\right)\right) + O\left(h^2\right),\end{aligned} \tag{47}$$

$$\begin{aligned}\frac{\partial^3 u}{\partial x^3}\left(x_i, t_j\right) = \frac{1}{2h^3}\left(u\left(x_{i+2}, t_j\right) - 2u\left(x_{i+1}, t_j\right) + 2u\left(x_{i-1}, t_j\right)\right.\\ \left.-u\left(x_{i-2}, t_j\right)\right) + O\left(h^2\right),\end{aligned} \tag{48}$$

$$\begin{aligned}\frac{\partial^3 u}{\partial x^3}\left(x_i, t_j\right) = \frac{1}{2h^3}\left(5u\left(x_i, t_j\right) - 18u\left(x_{i-1}, t_j\right) + 24u\left(x_{i-2}, t_j\right) - 14u\left(x_{i-3}, t_j\right)\right.\\ \left.+3u\left(x_{i-4}, t_j\right)\right) + O\left(h^2\right).\end{aligned} \tag{49}$$

Finally, we consider the interval $S_{i,j}$ such as for $\eta_j \in (t_{j-1}, t_j)$ we have $\partial^2 u/\partial t^2\left(x_i, \eta_j\right) \in S_{i,j}$. We propose to choose its endpoints as follows

$$\underline{S}_{i,j} = \min\left\{S_{i,j-1}^*, S_{i,j-1/2}^*, S_{i,j}^*\right\}, \ \overline{S}_{i,j} \approx \max\left\{S_{i,j-1}^*, S_{i,j-1/2}^*, S_{i,j}^*\right\}, \tag{50}$$

where

$$S_{i,j}^* = \frac{\partial^2 u}{\partial t^2}\left(x_i, t_j\right). \tag{51}$$

For the finite difference approximation of $\partial^2 u/\partial t^2$ we can use the formulas

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2}\left(x_i, t_j\right) = \frac{1}{k^2}\left(2u\left(x_i, t_j\right) - 5u\left(x_i, t_{j+1}\right) + 4u\left(x_i, t_{j+2}\right)\right.\\ \left.-u\left(x_i, t_{j+3}\right)\right) + O\left(k^2\right),\end{aligned} \tag{52}$$

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2}\left(x_i, t_j\right) = \frac{1}{12k^2}\left(-u\left(x_i, t_{j+2}\right) + 16u\left(x_i, t_{j+1}\right) - 30u\left(x_i, t_j\right)\right.\\ \left.+16u\left(x_i, t_{j-1}\right) - u\left(x_i, t_{j-2}\right)\right) + O\left(k^4\right),\end{aligned} \tag{53}$$

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2}\left(x_i, t_j\right) = \frac{1}{k^2}\left(2u\left(x_i, t_j\right) - 5u\left(x_i, t_{j-1}\right) + 4u\left(x_i, t_{j-2}\right)\right.\\ \left.-u\left(x_i, t_{j-3}\right)\right) + O\left(k^2\right).\end{aligned} \tag{54}$$

Remark 1. The stability criterion in the conventional backward finite difference method (see e.g. [10]) for solving the equation (1) with (2)-(3) is of the form

$$k \leq \min_{i=1,2,\ldots,n-1} \frac{h^2}{2\gamma\left(x_{i+1/2}\right)}. \tag{55}$$

The numerical experiments show that it is a good practice to choose the stepsize k such that the condition (55) is satisfied also for the interval scheme considered.

Remark 2. Note that in practice we have to approximate values of the unknown function $u = u(x,t)$ and its partial derivatives at (x_i, t_j) at some midpoints to obtain the endpoints of the intervals given in the error term (19) of the interval method (16)-(18) with (20). Let us first set the stepsizes h and k for the interval scheme considered above, such that the condition (55) is satisfied. Then we take $h = h/2$, $k = k/2$ and we solve (1) with (2)-(3) by the conventional backward finite difference method to get the approximations $u_{i,j}$ of $u(x_i, t_j)$. If we neglect the appropriate error terms given in (34)-(36), (40)-(42), (47)-(49), (52)-(54) and take $u_{i,j}$ instead of $u(x_i, t_j)$, then we can approximate values u needed in (29)-(30) and then the derivatives of u with the formulas of forward, central or backward finite differences depending on the position (i, j) in the grid.

Remark 3. In most cases we cannot find the endpoints of the intervals given in the error term of the interval method exactly. Hence, we cannot guarantee that the interval solutions obtained are such that they include the exact solution. The numerical tests on the initial-boundary value problems with the constant diffusion coefficient, for which the analytical solution can be derived, show that such inclusion takes place. If a diffusion coefficient γ is given as a function of x, then the exact solution of (1) with (2)-(3) is known only for some selected problems, i.e. for some special functions of diffusion coefficient and the appropriate initial and boundary conditions. Otherwise, the exact solution is unknown. Nevertheless, during the early stages of the interval method tune, we can compare the interval solutions obtained for different values of stepsizes h and k with themselves and also with the results obtained with other conventional methods for solving the problems considered (e.g. the finite element method implemented in ANSYS Multiphysics or COMSOL Multiphysics). We can also decrease values of the left endpoint and increase values of the right endpoint of intervals of the error term by some experimentally chosen percent value c (usually about 5% to 10%). In this way a probability that the error term interval includes the local truncation error at a given point is higher.

4 Numerical Experiment

Let us consider the equation of the form (1) given as follows

$$\frac{\partial u}{\partial t}(x,t) - \frac{\partial}{\partial x}\left(ax\frac{\partial u}{\partial x}(x,t)\right) = 0, \quad 0 < x < 1,\ t > 0. \tag{56}$$

If we take $a = 1$, then for the following initial and boundary conditions

$$u(x,0) = \exp(-x), \quad 0 \le x \le 1, \tag{57}$$

$$u(0,t) = 1/(t+1), \quad u(1,t) = 1/(t+1)\exp(-1/(t+1)), \quad t > 0, \tag{58}$$

the exact solution of (56) with (57)-(58) is of the form

$$u(x,t) = 1/(t+1)\exp(-x/(t+1)). \tag{59}$$

We set $T_{\max} = 1$. With the interval method applied in the floating-point interval arithmetic (see also [3], [4]) we get the interval solutions of the widths shown in Figure 1. Values of the exact solution and the interval solutions obtained for $h =$ 1E-2 and $k =$ 5E-3 are given in Table 1.

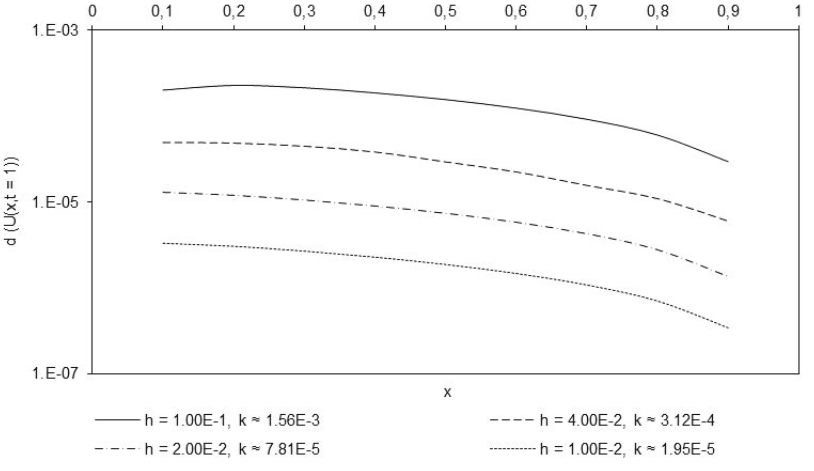


Fig. 1. Widths of the interval solution $U(x, t = 1)$ obtained with the interval method for the problem (56) with (57)-(58) for different values of h and k

Table 1. Values of the exact solution and the interval solutions obtained with the interval method, where $h =$ 1E-2 and $k =$ 5E-3

x	$u(x, t = 1)$	$U(x, t = 1)$
0.1	4.75614712250357000E-1	[4.7540584220522732E-1, 4.7579647018457326E-1]
0.2	4.52418709017979787E-1	[4.5227166459332143E-1, 4.5254690394948212 9E-1]
0.3	4.30353988212528904E-1	[4.3024427633830037 6E-1, 4.3044978785364283 8E-1]
0.4	4.09365376538990929E-1	[4.0928230308159498 7E-1, 4.0943801514511967 2E-1]
0.5	3.89400391535702434E-1	[3.8933788036115368 1E-1, 3.8945511477357780 2E-1]
0.6	3.70409110340858933E-1	[3.7036325344237530 6E-1, 3.7044929358354589 2E-1]
0.7	3.52344044859356717E-1	[3.5231213995900835 9E-1, 3.5237202463119238 4E-1]
0.8	3.35160023017819650E-1	[3.3514011042298575 3E-1, 3.3517749674445349 1E-1]
0.9	3.18814075810886647E-1	[3.1880468512077633 4E-1, 3.1882232017955273 7E-1]

5 Conclusions

The author presents the interval scheme for solving the diffusion equation with the diffusion coefficient given as a function of position and the boundary conditions of the first type. Moreover, some kind of the error term approximation of the conventional method is described. Since the endpoints of the error term intervals are approximated, the interval method considered just verifies the conventional method and we cannot guarantee that the exact solution belongs to the interval solutions obtained. Nevertheless, as the numerical experiments confirm, the exact solution does belong to the interval solutions obtained with the error term approximation proposed (see also Section 4).

References

1. Heitjans, P., Karger, J.: Diffusion in Condensed Matter – Methods, Materials, Models. Springer, Heidelberg (2005)
2. Crank, J.: The mathematics of Diffusion. Oxford University Press (1975)
3. Jankowska, M.A.: Remarks on Algorithms Implemented in Some C++ Libraries for Floating-Point Conversions and Interval Arithmetic. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009. LNCS, vol. 6068, pp. 436–445. Springer, Heidelberg (2010)
4. Jankowska, M.A.: C++ Library for Floating-Point Conversions, C++ Library for Floating-Point Interval Arithmetic, User and Reference Guides, Poznan University of Technology (2009), http://www.mjank.user.icpnet.pl/
5. Jankowska, M.A., Marciniak, A.: An Interval Finite Difference Method for Solving the One-Dimensional Heat Equation. LNCS (in press)
6. Jankowska, M.A.: An Interval Finite Difference Method of Crank-Nicolson Type for Solving the One-Dimensional Heat Conduction Equation with Mixed Boundary Conditions. In: Jónasson, K. (ed.) PARA 2010, Part II. LNCS, vol. 7134, pp. 157–167. Springer, Heidelberg (2012)
7. Manikonda, S., Berz, M., Makino, K.: High-order verified solutions of the 3D Laplace equation. WSEAS Transactions on Computers 4(11), 1604–1610 (2005)
8. Marciniak, A.: An Interval Version of the Crank-Nicolson Method – The First Approach. In: Jónasson, K. (ed.) PARA 2010, Part II. LNCS, vol. 7134, pp. 120–126. Springer, Heidelberg (2012)
9. Nagatou, K., Hashimoto, K., Nakao, M.T.: Numerical verification of stationary solutions for Navier-Stokes problems. Journal of Computational and Applied Mathematics 199(2), 445–451 (2007)
10. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press (2007)
11. Watanabe, Y., Yamamoto, N., Nakao, M.T.: A Numerical Verification Method of Solutions for the Navier-Stokes Equations. Reliable Computing 5(3), 347–357 (1999)

Arbitrary Precision Complex Interval Computations in C-XSC

Walter Krämer and Frithjof Blomquist

Bergische Universität Wuppertal, D-42119 Wuppertal, Germany
{kraemer,blomquist}@math.uni-wuppertal.de
http://math.uni-wuppertal.de/~xsc

Abstract. Based on the libraries MPFR and MPFI for arbitrary precision real and arbitrary precision real interval computations and corresponding interfaces to the C++ class library C-XSC, the new data type `MpfciClass` (multiple precision floating-point complex intervals) and corresponding operations/functions for arbitrary precision complex intervals have been implemented. Our new package allows to code mathematical expressions for the complex interval data type in their usual mathematical notation yielding easy to read and self-documenting source code. Meanwhile, more than 30 elementary mathematical functions have been realized. At any point of the program the user may change the precision setting of the computation. The maximum precision of complex interval variables is only restricted by memory limitations. Its exponent range is very large. To the knowledge of the authors there is no comparable package (with respect to the features provided) available worldwide. The new package is written in C++. It is freely available from `http://www2.math.uni-wuppertal.de/org/WRST/xsc/cxsc_software.html`.

Keywords: arbitrary precision, complex intervals, complex interval functions, C-XSC, MPFR, MPFI, reliable computations.

1 Introduction

Based on the ANSI C libraries MPFR [10] and MPFI [23] and corresponding interfaces [7, 5] to the C++ class library C-XSC [13–15] the new data type `MpfciClass` and corresponding operations/functions for arbitrary precision complex intervals have been implemented. Due to the C++ operator and function name overloading feature the new package may be used in a very comfortable way. Formulas and expressions may be written in their usual mathematical notation. The new package also supplies the user with many different predefined constructors allowing to mix usual numerical C-XSC data types (e.g. `real`, `interval`, `cinterval` etc.) with the new advanced arbitrary precision data types. Computations of e.g. a binary operation allow different precisions in the operands and a third precision setting for the resulting value. The user can modify/control the precision setting in a convenient way. All the usual elementary mathematical functions like the trigonometric functions, the inverse trigonometric functions, the

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 457–466, 2012.

hyperbolic and the inverse hyperbolic functions, logarithms and exponential functions, ... all allowing arbitrary precision complex intervals as input arguments, are provided by the package. The maximum precision of variables is only restricted by memory limitations of the computer system. The exponent range is very large. For example, the smallest value greater than 0 for the real part of an arbitrary precision complex interval is 2**(-1073741823), a really tiny number compared to the smallest positive double precision IEEE number. Arbitrary precision numbers are represented internally as binary numbers and the precision setting refers to the number of bits used for the mantissa. The package allows computations using say only 35 bit significands but also say 500402 bit significands, if these precision settings are demanded by the user.

Further details of the realization of the new package and of some of its predecessors are presented in [3, 20, 4–6].

2 Using the Ansi C Library MPFR from within C-XSC

The MPFR library [10] provides arbitrary precision real arithmetic operations as well as a rich set of mathematical functions. The results of all operations and functions are guaranteed to be best possible with respect to the current precision setting and the current rounding mode (both can be controlled by the user). The usage of the library is cumbersome and error prone. This may be seen e.g. by the following source code taken without modification from [11]:

```
#include <stdio.h>
#include <stdlib.h>
#include "mpfr.h"
int main (int argc, char *argv[])
{
  mp_prec_t p = atoi (argv[1]);
  mpfr_t a, b, c, d;
  mpfr_inits2 (p, a, b, c, d, (mpfr_ptr) 0);
  mpfr_set_str (a, "1e22", 10, GMP_RNDN);
  mpfr_sin (a, a, GMP_RNDN);
  mpfr_mul_ui (a, a, 173746, GMP_RNDN);
  mpfr_set_str (b, "17.1", 10, GMP_RNDN);
  mpfr_log (b, b, GMP_RNDN);
  mpfr_mul_ui (b, b, 94228, GMP_RNDN);
  mpfr_set_str (c, "0.42", 10, GMP_RNDN);
  mpfr_exp (c, c, GMP_RNDN);
  mpfr_mul_si (c, c, -78487, GMP_RNDN);
  mpfr_add (d, a, b, GMP_RNDN);
  mpfr_add (d, d, c, GMP_RNDN);
  mpfr_printf ("d = %1.16Re\n", d);
  mpfr_clears (a, b, c, d, NULL);
  return 0;
}
```

Here, the data type for arbitrary precision real numbers provided by the MPFR library is `mpfr_t`.

For most readers this code is probably far away from being self-documenting. You have to initialize all the arbitrary precision variables explicitly, you have to use cumbersome function calls for the binary arithmetic operations, you do not know which variables in a parameter list of a function call are input variables, which ones are output quantities and you have to free the variables at the end of your program explicitly. Do you immediately see which simple mathematical formula is to be evaluated?

Using our new interface `mpfrclass.hpp` to make MPFR available within C-XSC, a similar program may be coded as follows:

```
#include <iostream>
#include "mpfrclass.hpp"       //arbitrary precision arithmetic

using namespace std;
using namespace MPFR;
using namespace cxsc;

  //allow abbreviation MP for multiple precision real data type:
#define MP MpfrClass

int main() {
  cout.precision(20);    //decimal digits printed for mp numbers

  PrecisionType p;               //used to set current precision

  for (;;) {
    cout << "Precision: ";
    cin >> p;                    //read number of bits to be used
    cout << endl;
    if (p <= 1) break;

    MP::SetCurrPrecision(p);          //p bit arithmetic is used

    MP a("1E22"), b("17.1"), c("0.42"), d;

    d= 173746*sin(a) + 94228*ln(b) - 78487*exp(c);     //formula

    cout << "Precision used: " << d.GetPrecision()
         << "  d: " << d << endl;
  }
  return 0;
}
```

Now the mathematical notation (due to the C++ features of operation and function name overloading) of the formula makes it easy to see what is going on.

The comfortable interface to the MPFR library used in the preceding C-XSC program is realized as a C++ class called `MpfrClass`.

The demonstrated effect of program simplification is even more pronounced for source codes written to perform interval computations. It is worth to mention that the original program also runs without changes in our C-XSC environment (of course, the corresponding header file `mpfrclass.hpp` must be included to allow the usage of the MPFR library from within C-XSC).

3 Remarks on the Realization/Implementation of Complex Interval Functions

In this note we only deal with complex intervals given as rectangles in the complex plane with sides parallel to the axes. For a different approach based on discs in midpoint-radius representation see [18].

Our implementations of complex interval functions are based on the following considerations: To a given complex interval argument

$$Z = X + i \cdot Y; \quad X = [x_1, x_2], \quad Y = [y_1, y_2]; \tag{1}$$

with real intervals X, Y we want to compute for the complex function $w = f(z)$ with $z = x + iy, w = u(z) + iv(z) = u(x, y) + iv(x, y) \in \mathbb{C}$ a sharp complex interval enclosure of the range of function values

$$W = f(Z) := \{w \,|\, w = f(z) \text{ with } z \in Z\}. \tag{2}$$

In general, the image $W = f(Z)$ of Z is not a rectangle with sides parallel to the axes. Thus, the best possible complex interval enclosure $F(Z) = U(Z) + i \cdot V(Z)$ of W will be afflicted with (unavoidable) overestimation [16, p.16]. Figure 1 presents the typical situation. A method for the computation of the real interval enclosures U, V for the real and imaginary parts, respectively, can now be derived from the following theorem [2]:

With $z = x + i \cdot y \in \mathbb{C}$, $w = f(z) = u(x, y) + i \cdot v(x, y)$, $\quad z = x + i \cdot y \in Z \subset \mathbb{C}$ we have Let $f = u + i \cdot v : Z \subset G \longrightarrow \mathbb{C}$ holomorph in the region G. Then the real and imaginary part functions $u(x, y)$ and $v(x, y)$, resp. are harmonic functions taking on their minimum and maximum values on the boundary of Z.

Let us denote the extremal point leading to the minimum value by m and that leading to the maximum value by M. In general, the extremal points m, M are not the corner points of the input interval Z. Thus, in many cases only one coordinate of an extremal point will be a representable floating-point number. As an illustrative example we present in Figure 2 for the principal value of the inverse tangent function $\arctan(z)$ some input intervals Z where only the y-coordinate of the corresponding point m is a representable number [16].

Let us consider e.g. the input interval denoted by $Z_1 = [x_1, x_2] + i[y_1, y_2]$ (see Figure 2) in the first quadrant outside the unit circle. In this region for the real part function $u(x, y)$ of $\arctan(z)$ it holds

$$u(x, y) = \frac{1}{2} \arctan \frac{2x}{1 - x^2 - y^2} + \frac{\pi}{2}, \tag{3}$$

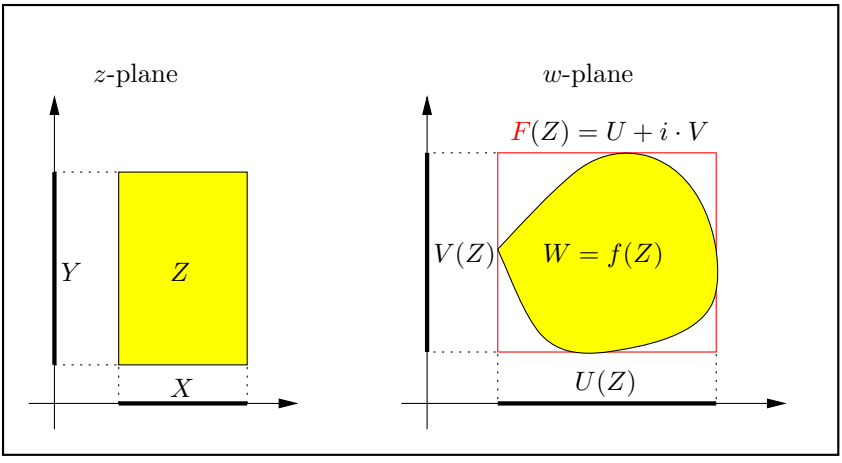


Fig. 1. Mapping by a complex interval function $F(Z) = F(X + iY)$

and if for the x-coordinate m_x of the point m we want to have $x_1 < m_x < x_2$, for a local minimizer m, the sufficient condition

$$\frac{\partial u(x,y)}{\partial x} = 0 \quad \left(\iff \quad y = +\sqrt{1+x^2} \quad \right)$$

must be fulfilled, yielding $y_1 = +\sqrt{1+m_x^2}$. Thus, the resulting x-coordinate is $m_x = \sqrt{y_1^2 - 1}$, which, in general, will not be a floating-point number. We have to compute a lower bound for $u(m) = u(\sqrt{y_1^2 - 1}, y_1)$ to find a lower bound for the minimum of the real part function $u(x, y)$ of $\arctan(z)$ over Z_1. Similar considerations apply for the maximum of $u(x, y)$ as well as for the minimum and maximum of the imaginary part function $v(x, y)$. To compute bounds for these values we evaluate the corresponding expressions (often in different parts of the complex plane different expressions for the same quantity must be used to avoid cancellation and/or significant overestimation) using arbitrary precision real arithmetic and/or arbitrary precision real interval arithmetic. Details may be found in [16, 17, 5]. For the real part function of the inverse cosine function $\arccos(z)$ the procedure is exemplarily presented in detail in [6].

4 Using Arbitrary Precision Complex Intervals from within C-XSC

To get familiar with the usage of arbitrary precision complex intervals let us discuss a very simple but illustrative example: Let us compute enclosures to different accuracies of the left hand side of the famous equation

$$e^{i\pi} + 1 = 0 \tag{4}$$

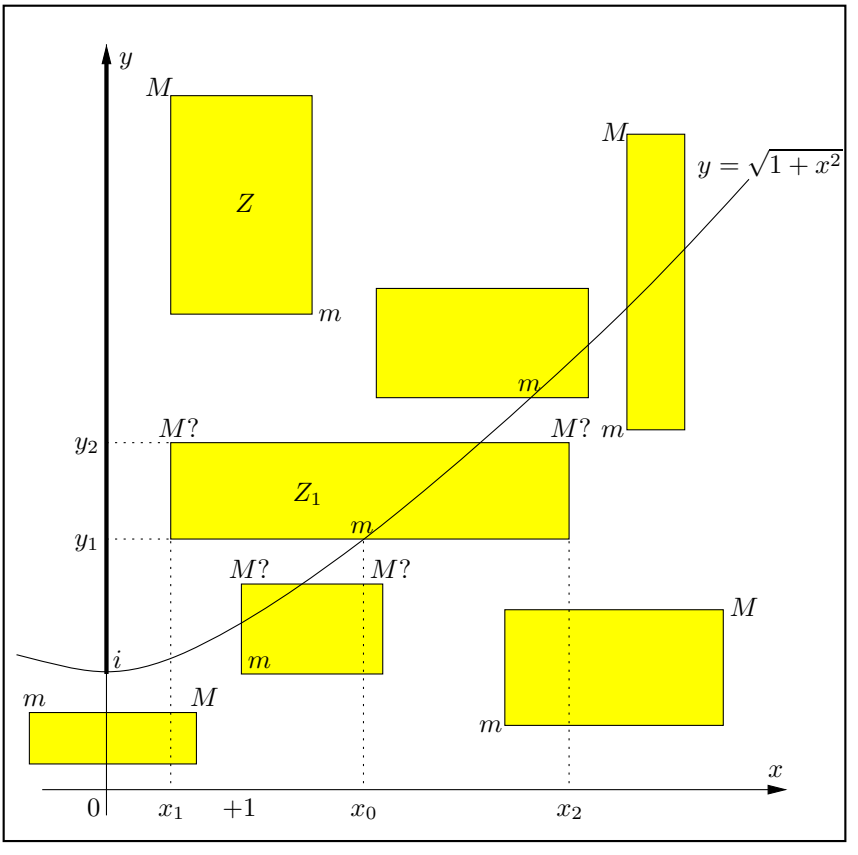


Fig. 2. Location in the z-plane of the extremal points m, M on the boundary of Z for the real part function $u(x, y)$ of $\arctan(z)$. $M?$ indicates that there are two possibilities, i.e. additional information is necessary to be able to decide which one is M.

using our arbitrary precison complex interval package and let us verify for different precision settings that 0 is always an element of the computed results.

The data type for arbitrary precision complex intervals in C-XSC is called `MpfciClass`. It is provided by the new interface file `mpfciclass.hpp`, where the acronym `mpfci` is an abbreviation for *multiple precision floating-point complex intervals*.

The constant π may be enclosed by calling the method `MpfiClass::Pi()`. This call returns (depending on the the current precision setting) an arbitrary precision real interval (data type `MpfiClass`) containing π. The accuracy of the enclosure is guaranteed to be best possible with respect to the current precision setting of the computation.

```
#include "mpfciclass.hpp" //arbitrary precision complex intervals

#define MP MpfciClass     //MP short form for that data type

using namespace std;
using namespace cxsc;
using namespace MPFI;     //arbitrary precision intervals

int main(){
   cout << endl << "*** Compute res = exp(i*pi) + 1 ... " << endl;
   cout << boolalpha;

   MP I(0,1); //imaginary unit
   MP res;    //result of formula evaluation
   bool zeroInResults=true;

   for (int p= 50; p<2E6; p+= p) {
     MP::SetCurrPrecision(p);
     cout << "Current precision setting: "
          << MP::GetCurrPrecision() << " bits" << endl;

     res= exp( I*MpfiClass::Pi() ) + 1;
     //it holds exp(i*pi) = -1, i.e. res must contain 0

     zeroInResults= zeroInResults && (0 <= res);

     cout << "  res: " << res << endl;
   }
   cout << "Last enclosure, diam( Re res + Im res ): "
        << diam( Re(res) + Im(res) ) << endl;
   cout << "0 element of all computed results: "
        << zeroInResults << endl;
   return 0;
}
```

Running the program produces the following (shortened) output:

```
*** Compute res = exp(i*pi) + 1 ...
Current precision setting: 50 bits
  res: ([0, 8.88179e-16], [-3.43025e-15, 1.22465e-16])
Current precision setting: 100 bits
  res: ([0, 7.88861e-31], [-2.98588e-30, 1.69569e-31])
Current precision setting: 200 bits
  res: ([0, 6.22302e-61], [-2.37501e-60, 1.14200e-61])
Current precision setting: 400 bits
  res: ([0, 3.87260e-121], [-1.48593e-120, 6.31080e-122])
                 ...                    ...
```

```
Current precision setting: 204800 bits
  res: ([0, 1.13996e-61651], [-4.50739e-61651, 5.24363e-61653])
Current precision setting: 409600 bits
  res: ([0, 1.29950e-123302], [-4.20965e-123303, 4.77704e-123302])
Current precision setting: 819200 bits
  res: ([0, 1.68870e-246604], [-3.30692e-246604, 3.44788e-246604])
Current precision setting: 1638400 bits
  res: ([0, 2.85171e-493208], [-6.45615e-493208, 4.95067e-493208])

Last enclosure, diam( Re res + Im res ): 1.42585e-493207
0 element of all computed results: true
```

The output shows - as expected - that 0 is always contained in the computed results. The operator `<=` allows to check whether the left hand side (here the point 0) is contained in the right hand side (here the arbitrary precision complex interval `res`). The last but one line of the output tells us that the diameter of the enclosure computed using a precision setting of about 1.6 million bits is smaller than $1.5 \cdot 10^{-493207}$. We can also see that doubling the current precision - as it is done in each step of the loop - results in doubling the accuracy of the result (the quality of the enclosure). Thus, from a numerical point of view, the results are very satisfactory.

5 Conclusion

K. Petras states in [21, 22] that for realistic parameters the numerical computation (using complex arithmetic) of an integral representation for arithmetic-average Asian options may lead to numerical cancellation of several hundred decimal digits due to highly oscillatory integrands, emphasizing the need for arbitrary precision complex interval packages.

Meanwhile, the possibility to work with arbitrary precision complex intervals is an outstanding supplement to C-XSC provided by our new class `MpfciClass`. To the knowledge of the authors there is no comparable software available world wide. Our implementation is heavily based on the MPFR and MPFI libraries. But both libraries do not offer direct support for complex function or complex interval function evaluations. Thus, the realization of complex interval functions is still costly. But meanwhile a rather complete set of mathematical functions are implemented (trigonometric, inverse trigonometric, hyperbolic, inverse hyperbolic, logarithms, exponentials, power functions, ...) all allowing arbitrary precision complex intervals as input arguments. Our package extends C-XSC smoothly. E.g., it provides a lot of constructors and allows to mix different numerical C-XSC data types when coding mathematical expressions. For more details see the extensive discussion in [5] (up to now only available in German). The implementation of our new package as well as many other C-XSC extensions are freely available from our website `http://www2.math.uni-wuppertal.de/org/WRST/xsc/cxsc_software.html`.

References

1. Adams, E., Kulisch, U.: Scientific Computing With Automatic Result Verification. Academic Press, Inc. (1993)
2. Behnke, H., Sommer, F.: Theorie der analytischen Funktionen einer komplexen Veränderlichen. Springer, Berlin (1962)
3. Blomquist, F., Hofschuster, W., Krämer, W., Neher, M.: Complex Interval Functions in C-XSC. Preprint BUW-WRSWT 2005/2, Bergische Universität Wuppertal, pp. 1–48 (2005)
4. Blomquist, F., Hofschuster, W., Krämer, W.: A Modified Staggered Correction Arithmetic with Enhanced Accuracy and Very Wide Exponent Range. In: Cuyt, A., Krämer, W., Luther, W., Markstein, P. (eds.) Numerical Validation. LNCS, vol. 5492, pp. 41–67. Springer, Heidelberg (2009)
5. Blomquist, F., Hofschuster, W., Krämer, W.: C-XSC-Langzahlarithmetiken für reelle und komplexe Intervalle basierend auf den Bibliotheken MPFR und MPFI. Preprint BUW-WRSWT 2011/1, Bergische Universität Wuppertal (2011), http://www2.math.uni-wuppertal.de/org/WRST/literatur/lit_wrswt.html#prep2011
6. Blomquist, F., Krämer, W.: Interval Enclosure of Re(arccos(Z)) for Complex Intervals Z. Preprint, University of Wuppertal (2011) (to appear)
7. Brand, H.-S.: Integration und Test einer Langzahlintervallbibliothek in C-XSC. Bachelor-Arbeit, Universität Wuppertal (2010)
8. Braune, K., Krämer, W.: High Accuracy Standard Functions for Real and Complex Intervals. In: Kaucher, E., Kulisch, U., Ullrich, C. (eds.) Computerarithmetic: Scientific Computation and Programming Languages, pp. 81–114. Teubner, Stuttgart (1987)
9. Braune, K.: Standard Functions for Real and Complex Point and Interval Arguments with Dynamic Accuracy. Computing Supplementum 6, 159–184 (1988)
10. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software (TOMS) 33(2) (June 2007)
11. Ghazi, K.R., Lefèvre, V., Théveny, P., Zimmermann, P.: Why and how to use arbitrary precision. Computing in Science and Engineering 12(3), 62–65 (2010)
12. Grimmer, M., Petras, K., Revol, N.: Multiple Precision Interval Packages: Comparing Different Approaches. In: Alt, R., Frommer, A., Kearfott, R.B., Luther, W. (eds.) Dagstuhl Seminar 2003. LNCS, vol. 2991, pp. 64–90. Springer, Heidelberg (2004)
13. Hofschuster, W., Krämer, W.: C-XSC 2.0 – A C++ Library for Extended Scientific Computing. In: Alt, R., Frommer, A., Kearfott, R.B., Luther, W. (eds.) Dagstuhl Seminar 2003. LNCS, vol. 2991, pp. 15–35. Springer, Heidelberg (2004)
14. Hofschuster, W., Krämer, W., Neher, M.: C-XSC and Closely Related Software Packages. In: Cuyt, A., Krämer, W., Luther, W., Markstein, P. (eds.) Numerical Validation. LNCS, vol. 5492, pp. 68–102. Springer, Heidelberg (2009)
15. Klatte, R., Kulisch, U., Wiethoff, A., Lawo, C., Rauch, M.: C-XSC - A C++ Class Library for Extended Scientific Computing. Springer, Heidelberg (1993)
16. Krämer, W.: Inverse Standardfunktionen für reelle und komplexe Intervallargumente mit a priori Fehlerabschätzungen für beliebige Datenformate, Dissertation, Universität Karlsruhe (1987)
17. Krämer, W.: Inverse Standard Functions for Real and Complex Point and Interval Arguments with Dynamic Accuracy. Computing Supplementum 6, 185–212 (1988)

18. Krier, R.: Komplexe Kreisarithmetik. PhD thesis, Universität Karlsruhe (1973)
19. Lohner, R.: Interval Arithmetic in Staggered Correction Format. In: [1], pp. 301–342 (1993)
20. Neher, M.: Complex Standard Functions and Their Implementation in the CoStLy Library. ACM Transactions on Mathematical Software 33(1), 27 pages (2007)
21. Petras, K.: A Method for Calculating the Complex Complementary Error Function with Prescribed Accuracy. Preprint, TU Braunschweig (2002), http://www-public.tu-bs.de:8080/~petras/publications.html
22. Petras, K.: Numerical Computation of an Integral Representation for Artihmetic-Average Asian Options. Preprint, TU Braunschweig (2002), http://www-public.tu-bs.de:8080/~petras/publications.html
23. Revol, N., Rouillier, F.: Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library. Reliable Computing 11, 275–290 (2005)

Tuning the Multithreaded Interval Method for Solving Underdetermined Systems of Nonlinear Equations

Bartłomiej Jacek Kubica

Institute of Control and Computation Engineering,
Warsaw University of Technology, Poland
bkubica@elka.pw.edu.pl

Abstract. The paper presents the author's investigations in the field of improving the performance of a multithreaded interval solver of equations systems, targeted for underdetermined systems. New heuristics to choose the interval Newton operator variant and bisection direction are proposed. Numerical results for several benchmark problems are presented and analyzed. Also some other tools and future possible improvements are suggested.

Keywords: underdetermined systems, nonlinear equations, interval computations, Newton operator, bisection, heuristic.

1 Introduction

Underdetermined systems of equations are systems of the form $f(x) = 0$, where $x \in [\underline{x}, \overline{x}]$ and $f: \mathbb{R}^n \to \mathbb{R}^m$, $m < n$. Such systems are encountered e.g. in robotics [16], stability theory of dynamical systems [20], differential equations solving [17] or seeking the Pareto set of a multicriteria problem [15]. Solution sets of such problems are not composed of isolated points, but are continuous manifolds and finding all such solutions is an ill-posed problem. Interval methods are a feasible approach to solve it as they can enclose the solution sets by a collection of boxes. We skip the basic of interval calculus, referring to several textbooks on the topic (e.g. [6], [8], [19]).

To the best knowledge of the author, the first paper considering applying interval methods to underdetermined problems was due to Neumaier [18]. The author developed a solver for this kind of problems and presented it in [11]. Several features of the solver were investigated in subsequent papers. In particular, shared-memory parallelization was considered [12], advanced uses of the Intel TBB [3] library [14] and issues of the efficient convergence [13].

This paper considers heuristics for selection of proper interval tools in the solver.

Motivation. Interval arithmetic provides several useful and powerful techniques to solve nonlinear equations. In particular we get:

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 467–476, 2012.

– various kinds of interval Newton operators (and different preconditioning strategies for them, if useful),
– various kinds of consistency operators,
– other constraint satisfaction and constraint propagation tools.

Also, these techniques can often be parametrized in a few different ways.

It is known that crucial for the efficient application of interval methods to a specific problem is developing a proper heuristic to choose appropriate interval tools and parametrize them properly. For example, Merlet proposes a bunch of sophisticated heuristics to solve robot-kinematics-related problems [16]. Independently, Goualard and Jermann propose some methods to select the variables for narrowing using the box-consistency operator [5].

For our problem, previous experiments (see [11], [13]) have shown that the componentwise Newton operator (Ncmp) [7] and Gauss-Seidel (GS) operator [8] perform the best, but their performance varies highly on different problems. Indeed, when we look on Tables 1 and 2 in Section 4, below, the componentwise operator outperforms GS e.g. for Rheinboldt and Puma6 problems, but is much worse for TwoCirc, Hippopede or Puma7. A clever heuristic to choose the operator or switch between them seems necessary to deal with practical problems efficiently.

2 Generic Algorithm

Before we consider the developed heuristics, let us present the basic meta-algorithm of the solver. It is the branch-and-prune (b&p) method that can be expressed as follows (notation from [9] is used):

```
IBP (x^(0); f)
// x^(0) is the initial box, f(·) is the interval extension of the function f: R^n → R^m
// L_ver is the list of boxes verified to contain a segment of the solution manifold
// L_pos is the list of boxes that possibly contain a segment of the solution manifold
L = L_ver = L_pos = ∅ ;
x = x^(0) ;
loop
    process the box x, using the rejection/reduction tests ;
    if (x does not contain solutions) then discard x ;
    else if (x is verified to contain a segment of solution manifold) then push (L_ver, x) ;
    else if (the tests resulted in two subboxes of x: x^(1) and x^(2)) then
        x = x^(1) ;
        push (L, x^(2)) ;
        cycle loop ;
    else if (x is small enough) then push (L_pos, x) ;
    if (x was discarded or stored) then
        x = pop (L) ;
        if (L was empty) then exit loop ;
    else
        bisect (x), obtaining x^(1) and x^(2);
        x = x^(1) ;
        push (L, x^(2)) ;
    end if ;
end loop
```

In previous papers, several rejection/reduction tests (mentioned in the above pseudocode) have been considered. According to the experiences of [11] and [13] we concentrate on two of them:

- the componentwise Newton operator with Herbort and Ratz heuristic to choose pairs equation-variable for reduction [7],
- the Gauss-Seidel operator with a rectangular matrix; Gauss elimination with full pivoting used for elements choice and inverse-midpoint preconditioning [8], [11], [13].

The componentwise operator can be expressed by the following formula:

$$N_{cmp}(\mathbf{x}, \mathsf{f}, i, j) = \operatorname{mid} \mathbf{x}_j - \frac{\mathsf{f}_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}, \operatorname{mid} \mathbf{x}_j, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_n)}{\frac{\partial \mathsf{f}_i}{\partial x_j}(\mathbf{x}_1, \ldots, \mathbf{x}_n)} \ , \tag{1}$$

while GS by:

$$GS(\mathbf{x}, \mathsf{f}, i, j) = \check{x}_i - \Big(Y_{i:} \cdot \mathsf{f}(\check{x}_1, \ldots, \check{x}_m, \mathbf{x}_{m+1}, \ldots, \mathbf{x}_n) \ + \\ + \sum_{j=1, j \neq i}^{m} Y_{i:} \cdot \mathbf{J}_{:j} \cdot (\mathbf{x}_j - \check{x}_j) \Big) / (Y_{i:} \cdot \mathbf{J}_{:i}) \ . \tag{2}$$

We denote the midpoint $\check{x}_i = \operatorname{mid} \mathbf{x}_i$ for brevity and the Jacobi matrix as $\mathbf{J}$ (while Y is the preconditioning matrix).

The differences between both operators can be summarized as follows:

- the componentwise operator linearizes the function wrt (with respect to) one variable only at a time, while GS – wrt all of them,
- the componentwise operator tries to narrow all variables using all equations (Herbort and Ratz heuristic [7]), while GS chooses m variables (out of n) and uses only one equation for each of them,
- the componentwise operator – because of its essence – does not require any preconditioning or other matrix operations.

In [11], we observed that for some problems the Ncmp operator is more efficient, while GS – for other ones, but no explanation of this phenomenon was given. Some hints were done later [13]. While the full explanation might be complex, our current understanding of the phenomenon allowed us to develop an efficient heuristic for choosing the operator, described below in Subsection 3.1.

3 Proposed Heuristics

In this paper the author concentrates on heuristics for two purposes: choosing the proper Newton operator variant and choosing proper variable for bisecting.

3.1 Choice of the Interval Newton Operator

From the current experience we can formulate the following observation: the componentwise operator seems to be more efficient for larger values of the accuracy parameter ε, while GS – for smaller ones. Why is it so?

Please note the formulae for both operators: (1) and (2). We can interpret them as follows: Ncmp uses the natural interval extension of $f(\cdot, \cdot, \ldots, \check{x}_j, \cdot, \ldots, \cdot)$, while GS – its centered form (see e.g. [6], [8]). It is a commonly known fact that centered forms are more efficient than natural interval extensions for small boxes (they converge quadratically with the interval's diameter, while natural extensions – converge linearly).

Ratschek and Rokne [19] proposed a simple heuristic: if the width of a box exceeds $0.5/n$ (where n is the box's dimension), then use the natural interval extension; otherwise – a centered form.

Applying this heuristic results in a simple condition to choose the Newton operator. However, preliminary experiments with checking if *all* components of the box have diameters below the threshold value were rather discouraging.

As the operator is supposed to contract domains of m variables, we use the Ncmp operator if at most $(n - m)$ of the variables have diameters below the threshold value and GS – otherwise. Please note, this reduces to checking all components for well-determined systems, i.e. when $n = m$.

3.2 Choice of the Variable for Bisection

The most common procedure to choose the box's component for bisection is to take the longest one. It is used in the developed solver, also. Some authors propose other policies to increase the efficiency, e.g. the maximal smear. Many of other policies have been considered in [4].

In our opinion, most of the previous approaches have been based on a wrong assumption – they tried to minimize the diameter of $\mathsf{f}(\cdot)$ on boxes resulting from bisection. But bisection is not the only and not the *main* tool used by the b&p procedure. So, the objective of bisection should be to result in boxes *that will be narrowed* by the Newton operator (or other rejection/reduction procedures) easily.

What does that mean? Let us consider a trivial example – a problem with one equation in two variables, e.g. $100 \cdot x_1 - x_2 = 0$ and the box $\mathbf{x} = [-2, 4] \times [-1, 1]$. How to bisect this box? Both, maximal diameter and maximal smear heuristics, would advice to bisect the first component that would result in boxes $[-2, 1] \times [-1, 1]$ (containing the whole solution set of $\mathbf{x}$) and $[1, 4] \times [-1, 1]$ (containing no solutions). Will it help the Newton operator? Such an operator will not be able to narrow the domain of second variable (there is a solution for each $x_2 \in [-1, 1]$) and if there were problems narrowing the first variable – they remain (because $\mathbf{x}_2$ was not narrowed)! However, if we bisect the second variable, obtaining $[-2, 4] \times [-1, 0]$ and $[-2, 4] \times [0, 1]$, any Newton operator should narrow the domain for $\mathbf{x}_1$ much easier than for the initial box.

So, choosing minimal smear seems to be a good strategy. Yet, if we stick to this policy, the b&p procedure might loose its convergence at all! To assure convergence (in case the Newton operator fails to narrow some boxes anyway) we have to bisect the longest component, if the difference between the components has become too high.

The resulting heuristic will be described now. It is a bit complex and some of its features have been adapted by a hit-and-miss process and could be improved, certainly. Yet, as we can see in Section 4, it performs very well.

`heuristic_choosing_a_variable_for_bisection`
find the index j_{max} and diameter w_{max} of the component with maximal diameter;
find the index j_{min} and diameter w_{min} of the component with minimal diameter;
find the index $j_{max\,unnarrowed}$ and diameter $w_{max\,unnarrowed}$ of the longest component *not reduced* by the last use of the Newton operator;
`if` (Newton reduced no components or $w_{max} > 1.5 \cdot w_{max\,unnarrowed}$) `then return` j_{max};
`else if` $(w_{max\,unnarrowed} > 8 \cdot w_{min})$ `then return` $j_{max\,unnarrowed}$;
find the index j and diameter w of the variable with smallest maximal magnitude of the Jacobi matrix in all rows;
`if` $(w > 0.1)$ `then return` j;
`else return` $j_{max\,unnarrowed}$;
`end heuristic_choosing_a_variable_for_bisection`

3.3 Other Tuning

Recent versions of the C-XSC library contain several improvements and allow several low-level optimizations [10]. In all considered examples, the ordinary floating-point arithmetic (no long accumulator or DotK algorithm) was used, by setting `opdotprec = 1`. Other optimizations (following a very interesting tutorial [10]) include:

- using approximate computations of the inverse-midpoint preconditioner,
- using BLAS and LAPACK routines,
- compiling the C-XSC library with high optimization level: `-O3 -finline-functions`.

4 Computational Experiments

Numerical experiments were performed on a computer with 16 cores, i.e. 8 Dual-Core AMD Opterons 8218 with 2.6GHz clock. The machine ran under control of a Fedora 15 Linux operating system.ATLAS 3.9.11 was installed there for BLAS libraries. The solver was implemented in C++, using C-XSC 2.5.1 library for interval computations. The GCC 4.6.0 compiler was used and TBB 3.0 update 7.

According to previous experiences (see [12], [14]), 8 parallel threads were used to decrease computation time. Please not that parallelization does not affect the number of iterations, but the execution time, only.

The following test problems were considered:

- TwoCirc [11], one equation in two variables, $\varepsilon = 10^{-5}$,
- Hippopede [11], [18], two equations in three variables, $\varepsilon = 10^{-7}$,
- Rheinboldt [11], [18], [20], five equations in eight variables, $\varepsilon = 10^{-1}$,
- Puma7 [11], the classical Puma problem without the last equation; 7 equations in 8 variables, $\varepsilon = 10^{-4}$,
- Puma6 [11], as Puma7, but without two last equations; 6 equations in 8 variables, $\varepsilon = 0.05$,
- Broyden8, Broyden12, a classical *well-determined* system used to see how proposed heuristics work for well-determined case, 8 (resp. 12) variables/ equations, $\varepsilon = 10^{-6}$.

The following notation is used in the tables:

- fun.evals, grad.evals, bisecs – numbers of functions evaluations, its gradients evaluations and boxes bisections,
- bis.Newt, del.Newt – numbers of boxes bisected/deleted by the Newton step,
- pos.boxes, verif.boxes – number of elements in the computed lists of boxes containing possible and verified solutions,
- Leb.pos., Leb.verif. – total Lebesgue measures of both sets,
- time – computation time in seconds.

For Table 6 we also add another row, called "speedup (GS)" – it is the speedup obtained with respect to GS (Table 2), as variant of the algorithm, using the GS operator can be considered a reference as being most popular.

Table 1. Results for algorithm variant using the Ncmp operator

problem	TwoCirc	Hippopede	Rheinboldt	Puma7	Puma6	Broyden8	Broyden12
fun.evals	801803	2699927388	139332688	154679461	11751927	1202377	427756353
grad.evals	401033	1152619116	25886660	60371164	4437504	369040	106687152
bisecs	198552	288139075	2580844	4311987	369791	22109	4423890
bis.Newt.	39	5	2297	8	0	955	21407
del.Newt.	0	112340464	486782	1421980	66128	13837	3111490
pos.boxes	191401	146307724	1948672	2197904	273448	627	72693
verif.boxes	7080	45988	21768	440	136	0	0
Leb.pos.	7e-6	4e-14	0.000159	4e-29	1e-8	1e-47	2e-70
Leb.verif.	0.506	0.0048	0.001677	5e-15	1e-13	0.0	0.0
time (sec.)	< 1	1108	55	69	5	1	269

5 Other Investigations

We also tried to improve the performance by using some other commonly known tools, specifically:

- linear relaxations and linear programming to narrow boxes, see e.g. [18],
- linear-programming preconditioners instead of inverse-midpoint ones [8],
- box-consistency enforcing procedures [5].

Table 2. Results for algorithm variant using the GS operator

problem	TwoCirc	Hippopede	Rheinboldt	Puma7	Puma6	Broyden8	Broyden12
fun.evals	6784	15430086	31653840	3539627	3525810	138144	19516368
grad.evals	8006	17730152	35359380	4022044	3911952	232456	31298580
bisecs	2576	4414539	3509425	279139	324883	12822	1238334
bis.Newt.	4	0	463	274	64	1705	65772
del.Newt.	0	972032	765767	72988	65224	2739	322256
prcnd.evals	6784	7715043	6330768	505661	587635	17268	1626364
pos.boxes	160	2243236	1946582	116076	188328	0	0
verif.boxes	1200	49240	56433	21440	7040	1	1
Leb.pos.	3e-9	2e-16	4e-5	4e-32	2e-9	0.0	0.0
Leb.verif.	0.6039	0.0028	1e-4	3e-11	4e-8	2e-90	1e-117
time (sec.)	< 1	25	120	11	10	1	213

Table 3. Results for algorithm variant using switching of both operators

problem	TwoCirc	Hippopede	Rheinboldt	Puma7	Puma6	Broyden8	Broyden12
fun.evals	5979	4031172	106971654	4129065	3971463	161708	29310470
grad.evals	6708	4632996	25870090	4598440	3536400	80840	9768708
bisecs	2092	1148235	2568787	321191	294679	4102	385608
bis.Newt.	39	5	2166	20	0	949	21420
del.Newt.	0	249784	708349	86228	60048	2123	248555
prcnd.evals	5528	2015068	1464963	578764	507672	1139	95595
pos.boxes	128	580680	1697860	140160	206032	0	0
verif.boxes	1061	16576	38351	20800	2736	1	1
Leb.pos.	2e-9	1e-16	0.000140	2e-32	2e-9	0.0	0.0
Leb.verif.	0.5471	0.0029	0.001676	7e-13	2e-11	9e-78	6e-96
time (sec.)	< 1	6	64	12	9	< 1	28

Preliminary experiments were very discouraging and are not presented here. A particular problem is that most present LP solvers are not multithreaded-safe. The solver available in the C-XSC library [1] contains (at least for version 2.5.0) several errors and did not tend to work at all. A popular GLPK 4.45 solver computed correct results, but its calls had to be executed in a critical section as it is not designed for multithreaded environments. Nevertheless, even for a single thread the execution time for algorithm variants using linear relaxations were longer than for the variant from Table 6; the number of gradients evaluations happened to be slower, though.

It seems these tools are not ready to cooperate with interval algorithms, yet – in particular multithreaded ones. Also box-consistency – contrary to results of e.g. [5] – did not prove useful; even for well-determined problems Broyden8 and Broyden12! This is going to be subject of further investigations.

Table 4. Results for algorithm variant using switching of both operators and the maximal smear coordinate bisection

problem	TwoCirc	Hippopede	Rheinboldt	Puma7	Puma6	Broyden8	Broyden12
fun.evals	18419	2052704408	n/a	42462885	13801087	178759	32294958
grad.evals	22347	2580554972	n/a	58244396	16011936	86640	10471716
bisecs	8607	644969291	n/a	4151399	1334327	4450	414563
bis.Newt.	35	9	n/a	28	0	963	21757
del.Newt.	0	194224012	n/a	986476	174048	2348	265587
prcnd.evals	17957	1026351616	n/a	6054944	2168328	1289	97834
pos.boxes	731	186228188	n/a	877744	702912	0	0
verif.boxes	3762	591496	n/a	25700	5048	1	1
Leb.pos.	1e-8	7e-15	n/a	3e-31	2e-9	0.0	0.0
Leb.verif.	0.6157	0.00023	n/a	9e-13	2e-11	2e-10	1e-95
time (sec.)	< 1	3476	> 19268	135	38	< 1	30

Table 5. Results for algorithm variant using switching of both operators and the heuristical bisection

problem	TwoCirc	Hippopede	Rheinboldt	Puma7	Puma6	Broyden8	Broyden12
fun.evals	5979	1184676	102257609	931861	3673215	96705	23364196
grad.evals	6708	1361164	24024215	976696	3236880	65712	8625492
bisecs	2092	329911	2365335	64635	269711	3153	337884
bis.Newt.	39	5	4264	32	0	953	21510
del.Newt.	0	69760	675343	19372	54280	1364	205614
prcnd.evals	5528	591820	1197803	121792	461464	1294	138663
pos.boxes	128	149952	1474309	19580	188208	0	0
verif.boxes	1061	21672	91553	12208	3744	1	1
Leb.pos.	2e-9	1e-17	0.000139	8e-33	2e-9	0.0	0.0
Leb.verif.	0.5471	0.0037	0.003043	3e-12	4e-11	5e-56	2e-114
time (sec.)	< 1	2	58	3	8	< 1	27

6 Analysis of the Results

The proposed heuristic for switching between Newton operators (Table 3) clearly resulted in an improvement of performance of the investigated solver. Times are slightly worse that for Ncmp (Table 1) for Rheinboldt and Puma6 problems, but they are never worse than for GS (Table 2). Moreover, for Hippopede and Broyden12 problems, we obtained a dramatic speedup and results much better than for any of the Newton operators used alone – either Ncmp or GS.

The heuristic for bisection (Table 5) did not result in speedup for TwoCirc problem and only in a minor one for Puma6 or Broyden, but – again – speedup for Hippopede and Puma7 was dramatic. This is in contrast with more classical maximum smear heuristic (Table 4) or the heuristic proposed in [4] (used in SONIC solver), for which we do not present the results as only for TwoCirc

Table 6. Results for algorithm variant as in Table 5, using approximate matrix inversion and highly tuned C-XSC

problem	TwoCirc	Hippopede	Rheinboldt	Puma7	Puma6	Broyden8	Broyden12
fun.evals	5981	1184684	102257609	931889	3673215	96705	23364196
grad.evals	6710	1361172	24024215	976724	3236880	65712	8625492
bisecs	2092	329911	2365335	64635	269711	3153	337884
bis.Newt.	39	5	4264	32	0	953	21510
del.Newt.	0	69760	675343	19372	54280	1364	205614
prcnd.evals	5530	591824	1197803	121796	461464	1294	138663
pos.boxes	128	149952	1474309	19580	188208	0	0
verif.boxes	1061	21672	91553	12208	3744	1	1
Leb.pos.	2e-9	1e-17	0.000139	8e-33	2e-9	0.0	0.0
Leb.verif.	0.5471	0.0037	0.003043	3e-12	4e-11	5e-56	2e-114
time (sec.)	< 1	1	51	1	4	< 1	21
speedup (GS)	n/a	25.0	2.35	11.00	2.5	n/a	10.14

problem any results have been obtained; for other ones too much memory was required – as also for the Rheinboldt problem in Table 4.

Using the approximate inverse matrix computations and other library tuning (Table 6), resulted yet in a significant speedup wrt Table 5. Differences in numbers of gradient evaluations, etc. are negligible, but present, so both tables are presented separately.

A comment about Ncmp and GS. Please note the results for Puma6 problem in Tables 1 and 2. The Ncmp operator requires more gradient evaluations (which is equivalent to more iterations, i.e. more boxes considered), yet its execution time is much shorter. This happens because Ncmp does not require costly matrix operations, in particular preconditioner computations. Further development of the C-XSC library (approximate matrix computations, smart use of BLAS and LAPACK, sparse matrix types, etc.) may reduce this effect. On the other hand, also for Ncmp there is a potential for improvement, e.g. parallel computing of $\mathsf{f}(\cdot)$ for different projections of a box; e.g. on a GPU.

7 Conclusions

Proposed heuristics resulted in a great improvement of the performance of investigated solver. They are based on two variants of the Newton operator and clever choosing of box's component for bisection; not some more sophisticated techniques, like box-consistency or linear programming on linear relaxations. The latter occurred to be inappropriate for multithreaded solvers; at least currently.

The possibility of using LP preconditioners and other common tools – and heuristics to choose between them – is going to be subject of further research.

References

1. C-XSC interval library, http://www.xsc.de
2. GLPK linear programming solver, http://www.gnu.org/software/glpk/
3. Intel Threading Building Blocks, http://www.threadingbuildingblocks.org
4. Beelitz, T., Bischof, C.H., Lang, B.: A hybrid subdivision strategy for result-verifying nonlinear solvers. Tech.rep. 04/8, Bergische Universitiät Wuppertal (2004)
5. Goualard, F., Jermann, C.: On the Selection of a Transversal to Solve Nonlinear Systems with Interval Arithmetic. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006, Part I. LNCS, vol. 3991, pp. 332–339. Springer, Heidelberg (2006)
6. Hansen, E., Walster, W.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (2004)
7. Herbort, S., Ratz, D.: Improving the efficiency of a nonlinear-system-solver using the componentwise Newton method. Tech.rep. 2/1997, Universität Karlsruhe (1997)
8. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
9. Kearfott, R.B., Nakao, M.T., Neumaier, A., Rump, S.M., Shary, S.P., van Hentenryck, P.: Standardized notation in interval analysis (2002), http://www.mat.univie.ac.at/~neum/software/int/notation.ps.gz
10. Krämer, W., Zimmer, M., Hofschuster, W.: Using C-XSC for High Performance Verified Computing. In: Jónasson, K. (ed.) PARA 2010, Part II. LNCS, vol. 7134, pp. 168–178. Springer, Heidelberg (2012)
11. Kubica, B.J.: Interval methods for solving underdetermined nonlinear equations systems. SCAN 2008 Proceedings. Reliable Computing 15(3), 207–217 (2011)
12. Kubica, B.J.: Shared-memory parallelization of an interval equations systems solver – comparison of tools. Prace Naukowe Politechniki Warszawskiej. Elektronika 169, 121–128 (2009)
13. Kubica, B.J.: Performance inversion of interval Newton narrowing operators. Prace Naukowe Politechniki Warszawskiej. Elektronika 169, 111–119 (2009)
14. Kubica, B.J.: Intel TBB as a tool for parallelization of an interval solver of nonlinear equations systems. Tech.rep. no 09-02, ICCE WUT (2009)
15. Kubica, B.J., Woźniak, A.: Using the Second-Order Information in Pareto-set Computations of a Multi-criteria Problem. In: Jónasson, K. (ed.) PARA 2010, Part II. LNCS, vol. 7134, pp. 137–147. Springer, Heidelberg (2012)
16. Merlet, J.-P.: Interval analysis for certified numerical solution of problems in robotics. International Journal of Applied Mathematics and Computer Science 19(3), 399–412 (2009)
17. Meyn, K.H.: Solution of underdetermined nonlinear equations by stationary iteration methods. Numerische Mathematik 42, 161–172 (1983)
18. Neumaier, A.: The enclosure of solutions of parameter-dependent systems of equations. In: Moore, R. (ed.) Reliability in Computing. Academic Press (1988)
19. Ratschek, H., Rokne, J.G.: Interval methods. In: Horst, R., Pardalos, P.M. (eds.) Handbook of Global Optimization. Kluwer (1995)
20. Rheinboldt, W.C.: Computation of critical boundaries on equilibrium manifolds. SIAM Journal of Numerical Analysis 19, 653–669 (1982)

Applying an Interval Method for a Four Agent Economy Analysis

Bartłomiej Jacek Kubica and Adam Woźniak

Institute of Control and Computation Engineering,
Warsaw University of Technology, Poland
bkubica@elka.pw.edu.pl, A.Wozniak@ia.pw.edu.pl

Abstract. This paper presents an application of interval methods for simulation of a policy game played by four main agents of economy: profit maximizing companies active on competitive market, the monopolistic trade union (aggregated wage setters), the government and the central bank. After reduction of companies as an active agent, the resulting three agents decision system is modeled as a Stackelberg game with the central bank as the leader and trade union and government as composite followers, aggregated by means of the Nash solution. The simulation of policy game is provided. A previously developed interval method is applied to find Nash points rigorously.

Keywords: interval computations, non-cooperative game, Nash equilibrium, economy modeling.

1 Introduction

The evaluation of policy before the implementation has a paramount significance for policy makers. The development of tools to make these evaluations is therefore important. In recent years, simulation modeling based on game theory has become a powerful tool to analyze hypothetical and actual policy changes. The key notion in the game theory is a game solution (equilibrium of the game). And finding this solution is not a simple problem. Interval methods [3], [5] are one of the approaches to solve it – see [7], [8].

In our previous work [8], a reliable and efficient interval method has been developed to find Nash solutions of non-cooperative games. From the another side, the second author has proposed a model of policy game played by four main agents of economy: perfectly rational profit maximizing companies active on competitive market aggregated to one, the monopolistic trade union (aggregated wage setters), the government and the central bank [11]. The present paper describes an application of our interval method to find the solution in this policy game.

2 The Model

Describing closed economy in [11] we adopted the Keynesian approach and derived the single period model of economy using general AD/AS model expanded

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 477–483, 2012.

with four decision makers (agents). Models of this class are frequently used for analysis of policy games, cf. [2]. The decision variables of our agents are the level of employment, L, for profit maximizing companies, the nominal wage W, for the monopolistic trade union, the budget deficit B, for the government and the supply of money M, for the central bank. When the decisions are made, the economy 'produces' in the equilibrium state, the output Y, and the rate of inflation Π. The assertion that companies are rational opens the possibility to eliminate from further consideration the activity of first agent and using instead 'stiff' equation describing their best response. Consequently the number of agents is reduced by one and after appropriate calculation we have the following explicit equations relating the output (in logs y) and rate of inflation Π to the wage (in logs w), budget deficit B and supply of money (in logs m):

$$y = \gamma \cdot \left(m + \frac{B}{\exp(m)} - w\right) + (1-\gamma) \cdot y_c \ , \tag{1}$$

$$\Pi = P - 1 = \exp(p) - 1 \ ,$$

$$p = (1-\gamma) \cdot \left(m + \frac{B}{\exp(m)}\right) + \gamma \cdot w - (1-\gamma) \cdot y_c \ , \tag{2}$$

where $\gamma \in (0,1)$ is the labor elasticity in short-run Cobb-Douglas production function $Y = L^{\gamma} K^{1-\gamma}$ and

$$y_c = \frac{\gamma}{1-\gamma} \cdot \ln(\gamma) + \ln K \ . \tag{3}$$

Basing on the above interactions in the economy, the primal (isolated) decision problems of the remaining, active agents: the trade union, the government and the central bank, are defined in [11] in the following way.

The decision problem of the trade union is a maximization of a weighted sum of real wage W/P, and terms measuring trade union threshold violation:

$$TU_p = (w-p) + \alpha_1 \cdot \min(y - y_{TU}, 0) - \alpha_2 \cdot \max(\Pi - \Pi_{TU}, 0) \ , \tag{4}$$

where $0 < \alpha_2 < \alpha_1 < 1$.

The signs of terms describing threshold violation are selected in such a way that the first term is smaller than zero when output is smaller than its threshold y_{TU} calculated by the trade union basing on accepted level of unemployment and second – is smaller than zero when inflation is larger than the threshold Π_{TU}. Using equations (1) and (2) we obtain the following decision problem of the trade union:

$$\begin{aligned} \text{find } w^o = \arg\max_{0 \le w \le e} TU = (1-\gamma) \cdot \left(w - m - \frac{B}{\exp(m)} + y_c\right) + \\ + \alpha_1 \cdot \min\left(\gamma \cdot \left(m + \frac{B}{\exp(m)} - w\right) + (1-\gamma) \cdot y_c - y_{TU}, 0\right) + \\ - \alpha_2 \cdot \max\left(\exp\left((1-\gamma) \cdot \left(m + \frac{B}{\exp(m)}\right) + \gamma w - (1-\gamma) \cdot y_c\right) + \right. \\ \left. - 1 - \Pi_{TU}, 0\right) \ . \end{aligned} \tag{5}$$

Similar considerations laid to objective function of the government:

$$G_p = \beta_1 \cdot \min(B - B_G, 0) + \beta_2 \cdot \min(y - y_G, 0) - \beta_3 \cdot \max(\Pi - \Pi_G, 0) \ , \quad (6)$$

where $\beta_1, \beta_2, \beta_3 > 0$ and the decision problem of this agent:

$$\begin{aligned} \text{find } B^o = \arg\max_{0 \le B \le B_m} G = \beta_1 \cdot \min(B - B_G, 0) + \quad (7) \\ +\beta_2 \cdot \min\Big(\gamma \cdot \big(m + \frac{B}{\exp(m)} - w\big) + (1-\gamma) \cdot y_c - y_G, 0\Big) + \\ -\beta_3 \cdot \max\Big(\exp\big((1-\gamma) \cdot (m + \frac{B}{\exp(m)}) + \gamma w - (1-\gamma) \cdot y_c\big) + \\ -1 - \Pi_G, 0\Big) \ . \end{aligned}$$

And as the objective function of the central bank, we adopted the function depending on terms measuring inflation and output (unemployment) target missing:

$$CB = -\max(\Pi - \Pi_{CB}, 0) + \delta \cdot \min(y - y_{CB}, 0) \ , \quad (8)$$

The decision problem of the central bank was stated as:

$$\begin{aligned} \text{find } m^o = \arg\max_{m^- \le m \le m^+} CB = \quad (9) \\ -\max\Big(\exp\big((1-\gamma) \cdot (m + \frac{B}{\exp(m)}) + \\ +\gamma w - (1-\gamma) \cdot y_c\big) - 1 - \Pi_{CB}, 0\Big) + \\ +\delta \cdot \min\Big(\gamma \cdot \big(m + \frac{B}{\exp(m)} - w\big) + (1-\gamma) \cdot y_c - y_{CB}, 0\Big) \ . \end{aligned}$$

This completes the descriptive part of model – description of agents' decision problems with their decision instruments and interactions joining them. Now we must model the rules (protocol) regularizing the behavior of agents. In other words, we must describe cooperation-coordination mechanism in this economy. We assume that the central bank is independent in its decisions and acts first announcing the chosen supply of money. It acts as the leader. Knowing the level of this external variable, the trade union and the government (followers) negotiate the level of wage and the budget deficit. As a result, Nash-equilibrium of this game [10], determined by the aggregate demand and the aggregate supply equations gives the employment, the output and the inflation. The essential tool in analyzing the described game (so called Stackelberg one) is response mapping of aggregate followers. Let (w^N, B^N) denotes the Nash equilibrium in union – government negotiation. Having full information about decision problems of followers, the central bank is potentially able to compute $r(m_j) = (w^N(m_j), B^N(m_j))$ for the given set $\{m_j\}$ of money supplies that is to estimate response function:

$$m \to R(m) = (w^N(m), B^N(m)) \ . \quad (10)$$

Now the decision problem of the central bank takes the form:

$$\begin{aligned} \text{find } m = \arg\max_{m^- \le m \le m^+} CBS(m) = \\ -\max\Big(\exp\big((1-\gamma)\cdot(m+\frac{B^N(m)}{\exp(m)}\big)+ \\ +\gamma w^N(m)-(1-\gamma)\cdot y_c\big)-1-\Pi_{CB},0\Big)+ \\ +\delta\cdot\min\Big(\gamma\cdot\big(m+\frac{B^N(m)}{\exp(m)}-w^N(m)\big)+(1-\gamma)\cdot y_c-y_{CB},0\Big)\ . \end{aligned} \tag{11}$$

R is a function, only when we assume that $\big(w^N(m), B^N(m)\big)$ is unique for every m. We do not have a formula for this function, but it can be computed by locating the Nash equilibrium computationally. This is obtained by the algorithm from [8], reminded in the following section.

3 The Algorithm

Let us recall the algorithm proposed in [8]. It is based on the branch-and-bound schema (see e.g. [3], [5]) and can be described by the following pseudocode:

```
seek_Nash_equilibria (x^(0); f_1, ..., f_n)
// x^(0) is the initial box,
// f_i(·) is the interval extension of the function f_i: R^N → R, i = 1, ldots, n
// L_sol is the list of boxes verified to contain a Nash point
L = ∅;
L_sol = ∅;
enqueue (L, x^(0));
while (L is nonempty)
    dequeue (L, x);
    process the box x, using the rejection/reduction tests;
    if (x does not contain a solution) then discard x;
    else if (x is verified to contain a solution) then enqueue (L_sol, x);
    else if ( diam(x) ≤ ε) then enqueue (L_sol, x);
    else
        bisect (x), obtaining x^(1) and x^(2);
        x = x^(1);
        enqueue (L, x^(2));
    end if;
end while
// Second phase
foreach (x in L_sol)
    if (x cannot contain a Nash equilibrium) then discard x;
end foreach
```

As the problems to solve are relatively simple and have a low dimensionality, no parallelization (as considered in [8]) is necessary.

4 Implementation

Numerical experiments were performed on a computer with 16 cores, i.e. 8 Dual-Core AMD Opterons 8218 with 2.6GHz clock. The machine ran under control of a Fedora 14 Linux operating system, with Glibc 2.13. The solver was implemented in C++, using C-XSC 2.5.0 library for interval computations. The GCC 4.5.1 compiler was used.

A few words have to be devoted to representation of players criteria. They are non-smooth – functions min() and max() are used by the criteria several times. Interval algorithms do not have to be modified significantly in this case (see e.g. [5]), but the automatic differentiation code from C-XSC library (files `hess_ari.hpp` and `hess_ari.cpp`) does not include functions min() and max().

Fortunately, already when doing the research described in [9] this problem has been solved – the automatic differentiations class has been modified to include proper operations. See Subsection 5.2 of [9] for more information.

5 Computational Experiments

Now, we present simulation results obtained using our interval method to find the Nash equilibrium for hypothetical closed economy with described four agents. We adopt the following values of parameters:

– for the production function: three values of gamma are considered: $\gamma = 0.65$, $\gamma = 0.7$, $\gamma = 0.75$ and the capital $K = 1251.7$;

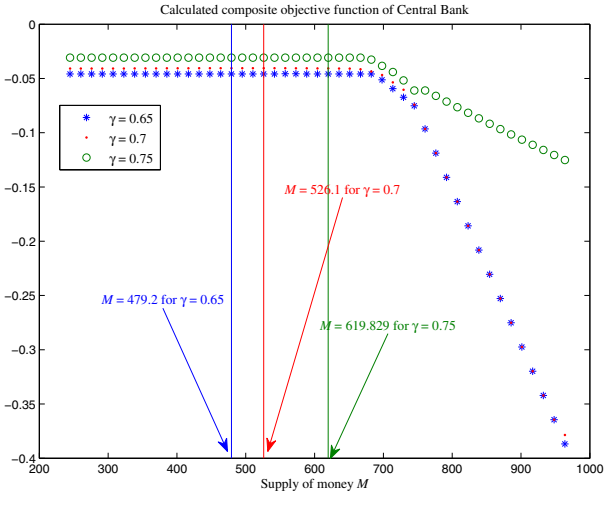


Fig. 1. Calculated composite objective functions of the central bank for different values of γ

- for primal decision problem of the trade union: weight of output $\alpha_1 = 0.6$, output threshold $y_{TU} = 6.55$, weight of inflation $\alpha_2 = 0.3$, inflation threshold $\Pi_{TU} = 0.1$ (a rather modern trade union);
- for primal decision problem of the government: deficit threshold $B_G = 40$, weight of deficit $\beta_1 = 10$, output threshold $y_G = 6.5$, weight of output $\beta_2 = 0.6$, weight of inflation $\beta_3 = 0.7$, inflation threshold $\Pi_G = 0.08$;
- for primal decision problem of the central bank: inflation threshold $\Pi_{CB} = 0.05$, output threshold $y_{CB} = 6.3$, weight of output $\delta = 0.2$ (a rather conservative central bank).

For each value of γ computations have been executed for 100 values of M – uniformly distributed from 244.7 to 1808.0. The computations lasted less than two seconds for each case. The shape of estimated composite function $CBS(m)$ is shown in Figure 1. Its maximal value is near zero and is realized by supply of money $M = 479.2$ for $\gamma = 0.65$, $M = 526.1$ for $\gamma = 0.7$ and $M = 619.8$ for $\gamma = 0.75$. Goals of the central bank are met – inflation is below $\Pi = 0.046$ for $\gamma = 0.65$, $\Pi = 0.040$ for $\gamma = 0.7$, $\Pi = 0.031$ for $\gamma = 0.75$, and the output $Y = 665.1$ (for all γ), is above threshold.

6 Conclusions and Future Work

The developed interval method for finding Nash equilibria proved to be useful in the simulation of a simple economical model. It will be very interesting to see if the method is efficient enough to handle a more complicated model that will describe an economy in a more realistic way. This is going to be subject of future research.

Other possible (and demanded) investigations include comparing results of the Stackelberg game with a Nash game [4], where there is no leader, but the Nash equilibrium of three players is localized. This would allow us to tell how beneficial it is to be the leader; does the information priority give the actual supremacy or not. Unfortunately, preliminary experiments occurred some problems due to the well-known cluster effect (the solution or solutions is approximated by a great deal of boxes that are difficult to analyze), so improving and tuning the solver seems to be necessary for this application.

References

1. C-XSC interval library, http://www.xsc.de
2. Acocella, N., Di Bartolomeo, G.: Non-neutrality of monetary policy in policy games. Working Paper no 49 (2002), Public Economics Department, University of Rome La Sapienza
3. Hansen, E., Walster, W.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (2004)
4. Jerger, J.: How strong is the case for a populist central banker? A note. European Economic Review 46, 623–632 (2002)

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 (2002), http://www.mat.univie.ac.at/~neum/software/int/notation.ps.gz
7. Kreinovich, V., Kubica, B.J.: From computing sets of optima, Pareto sets and sets of Nash equilibria to general decision-related set computations. Journal of Universal Computer Science 16, 2657–2685 (2010)
8. Kubica, B.J., Woźniak, A.: An Interval Method for Seeking the Nash Equilibria of Non-cooperative Games. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 446–455. Springer, Heidelberg (2010)
9. Kubica, B.J., Woźniak, A.: Using the Second-Order Information in Pareto-set Computations of a Multi-criteria Problem. In: Jónasson, K. (ed.) PARA 2010, Part II. LNCS, vol. 7134, pp. 137–147. Springer, Heidelberg (2012)
10. Nash, J.F.: Equilibrium points in n-person games. Proceedings of National Association of Science 36, 48–49 (1950)
11. Woźniak, A.: Policy Modeling in Four Agent Economy. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3038, pp. 615–622. Springer, Heidelberg (2004)

An Axiomatic Approach to Computer Arithmetic with an Appendix on Interval Hardware

Ulrich Kulisch

Institut für Angewandte und Numerische Mathematik
Karlsruher Institut für Technologie
Kaiserstrasse 12
D-76128 Karlsruhe Germany
Ulrich.Kulisch@kit.edu

Abstract. Different kinds of computer arithmetic follow an abstract mathematical pattern and are just special realizations of it. The basic mathematical concepts are sketched here. The concepts of rounding, of a screen, and of rounded arithmetic operations are defined in an axiomatic manner fully independent of special data formats and encodings. These abstract concepts are then applied and illustrated by the two elementary models of computer arithmetic for real numbers and for real intervals. In the latter case definition of the arithmetic operations as set operations does not suffice. Executable formulas have to be derived. We also demonstrate how this can be achieved. In an appendix we sketch a hardware realization of interval arithmetic and show that most of what is needed for it is already available on current x86-processors.

1 Introduction

A standard for floating-point arithmetic IEEE 754 was established in 1985. A revision was published in 2008. A standard for interval arithmetic is just under development. While these two standards have to consider data formats, encodings, implementation, exception handling, and so on this study concentrates on the mathematics of computer arithmetic. As usual mathematics is suited to focus the understanding on the essentials.

Frequently mathematics is seen as the science of structures. Analysis carries three kinds of structures: an algebraic structure, an order structure, and a topological or metric structure. These are coupled by certain compatibility properties, as for instance: $a \leq b \Rightarrow a + c \leq b + c$.

It is well known that floating-point numbers and floating-point arithmetic do not obey the rules of the real numbers $\mathbb{R}$. However, the rounding is a monotone function. So the changes to the order structure are minimal. This is the reason why the order structure plays a key role for an axiomatic approach to rounded computations.

This paper is restricted to the two elementary models that are covered by the two IEEE arithmetic standards, computer arithmetic on the reals and real

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 484–495, 2012.

intervals. Abstract settings of computer arithmetic for higher dimensional spaces like complex numbers, vectors and matrices for real, complex, and interval data can be developed following similar schemes. For more details see [6] and the literature listed there.

2 Computer Arithmetic Axioms

We begin this study by listing a few well-known concepts and properties of ordered sets.

Definition 1. *A relation* $\leq$ *in a set* M *is called an* order relation*, and* $\{M, \leq\}$ *is called an* ordered set[1] *if for all* $a, b, c \in M$ *the following properties hold:*

(O1) $a \leq a$ (reflexivity),
(O2) $a \leq b \ \wedge \ b \leq c \ \Rightarrow \ a \leq c$ (transitivity),
(O3) $a \leq b \ \wedge \ b \leq a \ \Rightarrow \ a = b$ (antisymmetry).

An ordered set M *is called* linearly *or* totally ordered *if in addition*

(O4) $a \leq b \vee b \leq a$ *for all* $a, b \in M$ (linearly ordered).

An ordered set M *is called*

(O5) *a* lattice *if for any two elements* $a, b \in M$*, the* $\inf\{a, b\}$ *and the* $\sup\{a, b\}$ *exist.*

(O6) *It is called* conditional completely ordered *if for every bounded subset* $S \subseteq M$*, the* $\inf S$ *and the* $\sup S$ *exist.*

(O7) *An ordered set* M *is called* completely ordered *or a* complete lattice *if for every subset* $S \subseteq M$*, the* $\inf S$ *and the* $\sup S$ *exist.*

With these concepts the real numbers $\mathbb{R}$ are defined as a conditional complete linearly ordered field.

In the definition of a complete lattice, the case $S = M$ is included. Therefore, $\inf M$ and $\sup M$ exist. Since they are elements of M, every complete lattice has a least and a greatest element.

If a subset $S \subseteq M$ of a complete lattice $\{M, \leq\}$ is also a complete lattice, $\{S, \leq\}$ is called a *complete sublattice* of $\{M, \leq\}$ if the two lattice operations inf and sup in both sets lead to the same result, i.e., if

$$\text{for all} \quad A \subseteq S, \quad \inf_M A = \inf_S A \quad \text{and} \quad \sup_M A = \sup_S A.$$

Definition 2. *A subset* S *of a complete lattice* $\{M, \leq\}$ *is called a* screen *of* M*, if every element* $a \in M$ *has upper and lower bounds in* S *and the set of all upper and lower bounds of* $a \in M$ *has a least and a greatest element in* S *respectively. If a minus operator exists in* M*, a screen is called* symmetric*, if for all* $a \in S$ *also* $-a \in S$*.*

[1] Occasionally called a partially ordered set.

As a consequence of this definition a complete lattice and a screen have the same least and greatest element. It can be shown that a screen is a complete sublattice of $\{M, \leq\}$ with the same least and greatest element, [6].

Definition 3. *A mapping* $\square : M \to S$ *of a complete lattice* M *onto a screen* S *is called a* rounding *if (R1) and (R2) hold:*

(R1) *for all* $a \in S$, $\square a := a$ (projection).
(R2) $a \leq b \Rightarrow \square a \leq \square b$ (monotone).

A rounding is called downwardly directed *resp.* upwardly directed *if for all* $a \in M$

(R3) $\square a \leq a$ *resp.* $a \leq \square a$ (directed).

If a minus operator is defined in M*, a rounding is called* antisymmetric *if*

(R4) $\square(-a) = -\square a$, *for all* $a \in M$ (antisymmetric).

The monotone downwardly resp. upwardly directed roundings of a complete lattice onto a screen are unique [6].

Definition 4. *Let* $\{M, \leq\}$ *be a complete lattice and* $\circ : M \times M \to M$ *a binary arithmetic operation in* M*. If* S *is a screen of* M*, then a rounding* $\square : M \to S$ *can be used to approximate the operation* $\circ$ *in* S *by*

$$\text{(RG)} \quad a \boxdot b := \square(a \circ b), \text{ for } a, b \in S.$$

If a minus operator is defined in M *and* S *is a symmetric screen of* M*, then a mapping* $\square : M \to S$ *with the properties (R1,2,4) and (RG) is called a* semimorphism[2].

Semimorphisms with antisymmetric roundings are particularly suited transferring properties of the structure in M to the subset S. It can be shown that semimorphisms leave a number of reasonable properties of ordered algebraic structures (ordered field, ordered vector space) invariant.

If an element $x \in M$ is bounded by $a \leq x \leq b$ with $a, b \in S$, then by (R1) and (R2) the rounded image $\square x$ is bounded by the same elements: $a \leq \square x \leq b$, i.e., $\square x$ is either the least upper (supremum) or the greatest lower (infimum) bound of x in S. Similarly, if for $x, y \in S$ the result of an operation $x \circ y$ is bounded by $a \leq x \circ y \leq b$ with $a, b \in S$, then by (R1), (R2), and (RG) also $a \leq x \boxdot y \leq b$, i.e., $x \boxdot y$ is either the least upper or the greatest lower bound of $x \circ y$ in S. If the rounding is upwardly or downwardly directed the result is the least upper or the greatest lower bound respectively.

[2] The properties (R1,2,4) and (RG) of a semimorphism can be shown to be necessary conditions for a homomorphism between ordered algebraic structures. For more details see [6].

3 Two Elementary Models

3.1 Floating-Point Arithmetic

The set $\{\mathbb{R}^*, \leq\}$ with $\mathbb{R}^* := \mathbb{R} \cup \{-\infty, +\infty\}$ is a complete lattice. Let $\mathbb{F}$ denote the set of finite floating-point numbers and $\mathbb{F}^* := \mathbb{F} \cup \{-\infty, +\infty\}$. Then $\mathbb{F}^*$ is a screen of $\{\mathbb{R}^*, \leq\}$. The least element of the set $\mathbb{R}^*$ and of the subset $\mathbb{F}^*$ is $-\infty$ and the greatest element is $+\infty$.

Definition 5. *With a rounding* $\square : \mathbb{R}^* \to \mathbb{F}^*$ *arithmetic operations* $\boxdot$ *in* $\mathbb{F}^*$ *are defined by*

$$\text{(RG)} \quad a \boxdot b := \square (a \circ b), \text{ for } a, b \in \mathbb{F}^* \text{ and } \circ \in \{+, -, *, /\},$$

with $b \neq 0$ *in case of division.*[3]

If a and b are adjacent floating-point numbers and $x \in \mathbb{R}$ with $a \leq x \leq b$, then because of (R1) and (R2) also $a \leq \square x \leq b$, i.e., there is never an element of $\mathbb{F}$ between an element $x \in \mathbb{R}$ and its rounded image $\square x$. The same property holds for the operations defined by (RG): If for $x, y \in \mathbb{F}$, $a \leq x \circ y \leq b$ then by (R1), (R2), and (RG) also $a \leq x \boxdot y \leq b$, for all $\circ \in \{+, -, *, /\}$, i.e., $x \boxdot y$ is either the greatest lower or the least upper bound of $x \circ y$ in $\mathbb{F}$.

Frequently used roundings $\square : \mathbb{R}^* \to \mathbb{F}^*$ are antisymmetric. Examples are the rounding to the nearest floating-point number, the rounding toward zero, or the rounding away from zero. A semimorphism transfers a number of useful properties of the real numbers to the floating-point numbers. The mathematical structure of $\mathbb{F}$ even can be defined as properties of $\mathbb{R}$ which are invariant with respect to semimorphism, [6].

For the monotone downwardly resp. upwardly directed roundings of $\mathbb{R}^*$ onto $\mathbb{F}^*$ often the special symbols $\triangledown$ resp. $\triangle$ are used. These roundings are not antisymmetric. They are related by the property:

$$\triangledown (-a) = - \triangle a \quad \text{and} \quad \triangle (-a) = - \triangledown a. \tag{1}$$

Arithmetic operations defined by (RG) and these roundings are denoted by $\boxed{\triangledown}$ and $\boxed{\triangle}$, respectively, for $\circ \in \{+, -, *, /\}$. These are heavily used in interval arithmetic.

3.2 Interval Arithmetic

Interval arithmetic over the real numbers deals with closed[4] and connected sets of real numbers. An interval is denoted by an ordered pair $[a_1, a_2]$. The first element is the lower bound and the second is the upper bound. An interval can be bounded or unbounded. If a bound is $-\infty$ or $+\infty$ the bound is not an

[3] In real analysis division by zero is not defined. It does not lead to a real number.
[4] A set of real numbers is called closed, if its complement is open.

element of the interval. Such intervals may also be written as $(-\infty, a]$ or $[b, +\infty)$ or $(-\infty, +\infty)$. The set of bounded real intervals is denoted by $\mathbb{IR}$. The set of bounded and unbounded real intervals is denoted by $\overline{\mathbb{IR}}$. With respect to the subset relation as an order relation the set of real intervals $\{\overline{\mathbb{IR}}, \subseteq\}$ is a complete lattice. The subset of $\overline{\mathbb{IR}}$ where all finite bounds are floating-point numbers of $\mathbb{F}$ is denoted by $\overline{\mathbb{IF}}$. $\{\overline{\mathbb{IF}}, \subseteq\}$ is a screen of $\{\overline{\mathbb{IR}}, \subseteq\}$. In both sets $\overline{\mathbb{IR}}$ and $\overline{\mathbb{IF}}$ the infimum of a subset of $\overline{\mathbb{IF}}$ is the intersection and the supremum is the interval hull[5]. The least element of both sets $\overline{\mathbb{IR}}$ and $\overline{\mathbb{IF}}$ is the empty set $\varnothing$ and the greatest element is the set $\mathbb{R} = (-\infty, +\infty)$.

The most frequently used rounding of $\overline{\mathbb{IR}}$ onto $\overline{\mathbb{IF}}$ is the upwardly directed rounding denoted by $\diamondsuit : \overline{\mathbb{IR}} \to \overline{\mathbb{IF}}$. It is characterized by the following properties:

(R1) $\diamondsuit\, \boldsymbol{a} = \boldsymbol{a}$, for all $\boldsymbol{a} \in \overline{\mathbb{IF}}$ (projection),
(R2) $\boldsymbol{a} \subseteq \boldsymbol{b} \Rightarrow \diamondsuit\, \boldsymbol{a} \subseteq \diamondsuit\, \boldsymbol{b}$, for $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IR}}$ (monotone),
(R3) $\boldsymbol{a} \subseteq \diamondsuit\, \boldsymbol{a}$, for all $\boldsymbol{a} \in \overline{\mathbb{IR}}$ (upwardly directed).

The result of the monotone upwardly directed rounding $\diamondsuit$ is unique. For $\boldsymbol{a} = [a_1, a_2] \in \overline{\mathbb{IR}}$ it is $\diamondsuit\, \boldsymbol{a} = [\, \triangledown a_1,\ \triangle a_2]$.

An interval $\boldsymbol{a} = [a_1, a_2]$ is frequently interpreted as a point in $\mathbb{R}^2$. This very naturally induces the order relation $\leq$ of $\mathbb{R}^2$ to the set of intervals $\overline{\mathbb{IR}}$. For two intervals $\boldsymbol{a} = [a_1, a_2]$ and $\boldsymbol{b} = [b_1, b_2]$ the relation $\leq$ is defined by $\boldsymbol{a} \leq \boldsymbol{b} :\Leftrightarrow a_1 \leq b_1 \wedge a_2 \leq b_2$.

For the $\leq$ relation for intervals compatibility properties hold between the algebraic structure and the order structure in great similarity to the real numbers. For instance:

(OD1) $\boldsymbol{a} \leq \boldsymbol{b} \Rightarrow \boldsymbol{a} + \boldsymbol{c} \leq \boldsymbol{b} + \boldsymbol{c}$, for all $\boldsymbol{c}$.
(OD2) $\boldsymbol{a} \leq \boldsymbol{b} \Rightarrow -\boldsymbol{b} \leq -\boldsymbol{a}$.
(OD3) $[0,0] \leq \boldsymbol{a} \leq \boldsymbol{b} \wedge \boldsymbol{c} \geq [0,0] \Rightarrow \boldsymbol{a} * \boldsymbol{c} \leq \boldsymbol{b} * \boldsymbol{c}$.
(OD4) $[0,0] < \boldsymbol{a} \leq \boldsymbol{b} \wedge \boldsymbol{c} > [0,0] \Rightarrow [0,0] < \boldsymbol{a}/\boldsymbol{c} \leq \boldsymbol{b}/\boldsymbol{c} \wedge \boldsymbol{c}/\boldsymbol{a} \geq \boldsymbol{c}/\boldsymbol{b} > [0,0]$.

Definition 6. *For intervals $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IR}}$ arithmetic operations $\circ \in \{+, -, *, /\}$ are defined as set operations*

$$\boldsymbol{a} \circ \boldsymbol{b} := \{a \circ b \mid a \in \boldsymbol{a} \wedge b \in \boldsymbol{b}\}. \tag{2}$$

Here for division we assume that $\boldsymbol{b} \neq [0,0]$. $\boldsymbol{a}/[0,0]$ is defined to be the empty set $\varnothing$ for any $\boldsymbol{a} \in \overline{\mathbb{IR}}$.

Using (1) it is easy to see that the monotone upwardly directed rounding $\diamondsuit : \overline{\mathbb{IR}} \to \overline{\mathbb{IF}}$ is antisymmetric, i.e.,

(R4) $\diamondsuit\, (-\boldsymbol{a}) = -\diamondsuit\, \boldsymbol{a}$, for all $\boldsymbol{a} \in \overline{\mathbb{IR}}$ (antisymmetric).

Arithmetic operations in $\overline{\mathbb{IR}}$ are inclusion isotone, i.e., $\boldsymbol{a} \subseteq \boldsymbol{b} \Rightarrow \boldsymbol{a} \circ \boldsymbol{c} \subseteq \boldsymbol{b} \circ \boldsymbol{c}$ or equivalently

[5] Which is the convex hull in the one dimensional case.

(OD5) $\boldsymbol{a} \subseteq \boldsymbol{b} \wedge \boldsymbol{c} \subseteq \boldsymbol{d} \Rightarrow \boldsymbol{a} \circ \boldsymbol{c} \subseteq \boldsymbol{b} \circ \boldsymbol{d}$, for all $\circ \in \{+, -, *, /\}$ (inclusion isotone).

For $\boldsymbol{a} = [a_1, a_2] \in \overline{\mathbb{IR}}$ we obtain by (2) immediately

$$-\boldsymbol{a} := [6](-1)*\boldsymbol{a} = [-a_2, -a_1]. \tag{3}$$

Definition 7. *With the upwardly directed rounding* $\diamondsuit : \overline{\mathbb{IR}} \to \overline{\mathbb{IF}}$ *binary arithmetic operations in* $\overline{\mathbb{IF}}$ *are defined by semimorphism:*

(RG) $\boldsymbol{a} \diamondsuit \boldsymbol{b} := \diamondsuit (\boldsymbol{a} \circ \boldsymbol{b})$, *for all* $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IF}}$ *and all* $\circ \in \{+, -, *, /\}$.

Here for division we assume that $\boldsymbol{a}/\boldsymbol{b}$ *is defined.*

If an interval $\boldsymbol{a} \in \overline{\mathbb{IF}}$ is an upper bound of an interval $\boldsymbol{x} \in \overline{\mathbb{IR}}$, i.e., $\boldsymbol{x} \subseteq \boldsymbol{a}$, then by (R1), (R2), and (R3) also $\boldsymbol{x} \subseteq \diamondsuit \boldsymbol{x} \subseteq \boldsymbol{a}$. This means $\diamondsuit \boldsymbol{x}$ is the least upper bound, the supremum of $\boldsymbol{x}$ in $\overline{\mathbb{IF}}$. Similarly if for $\boldsymbol{x}, \boldsymbol{y} \in \overline{\mathbb{IF}}$, $\boldsymbol{x} \circ \boldsymbol{y} \subseteq \boldsymbol{a}$ with $\boldsymbol{a} \in \overline{\mathbb{IF}}$, then by (R1), (R2), (R3), and (RG) also $\boldsymbol{x} \circ \boldsymbol{y} \subseteq \boldsymbol{x} \diamondsuit \boldsymbol{y} \subseteq \boldsymbol{a}$, i.e., $\boldsymbol{x} \diamondsuit \boldsymbol{y}$ is the least upper bound, the supremum of $\boldsymbol{x} \circ \boldsymbol{y}$ in $\overline{\mathbb{IF}}$. Occasionally the supremum $\boldsymbol{x} \diamondsuit \boldsymbol{y}$ of the result $\boldsymbol{x} \circ \boldsymbol{y} \in \overline{\mathbb{IR}}$ is called the tightest enclosure of $\boldsymbol{x} \circ \boldsymbol{y}$.

Arithmetic operations in $\overline{\mathbb{IF}}$ are inclusion isotone, i.e.,

(OD5) $\boldsymbol{a} \subseteq \boldsymbol{b} \wedge \boldsymbol{c} \subseteq \boldsymbol{d} \Rightarrow \boldsymbol{a} \diamondsuit \boldsymbol{c} \subseteq \boldsymbol{b} \diamondsuit \boldsymbol{d}$, for all $\circ \in \{+, -, *, /\}$ (inclusion isotone).

This is a consequence of the inclusion isotony of the arithmetic operations in $\overline{\mathbb{IR}}$, of (R2) and of (RG).

Because of the definition of the operations $\boldsymbol{x} \circ \boldsymbol{y}$ in $\overline{\mathbb{IR}}$ as set operations (2) the operations $\boldsymbol{x} \diamondsuit \boldsymbol{y}$ for intervals of $\overline{\mathbb{IF}}$ defined by (RG) are not practicable. The step from the definition of interval arithmetic by set operations to computer executable operations still requires some effort. This holds in particular for product sets like complex intervals and intervals of vectors and matrices, [6]. We sketch it here for the most simple case of floating-point intervals.

3.3 Executable Interval Arithmetic

For bounded, nonempty real intervals $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{IR}$ arithmetic operations are defined as set operations by $\boldsymbol{a} \circ \boldsymbol{b} := \{a \circ b \mid a \in \boldsymbol{a} \wedge b \in \boldsymbol{b}\}$. Here for $0 \notin \boldsymbol{b}$, $a \circ b$ is a continuous function of both variables. $\boldsymbol{a}$ and $\boldsymbol{b}$ are closed intervals. The product set $\boldsymbol{a} \times \boldsymbol{b}$ is a simply connected, bounded and closed subset of $\mathbb{R}^2$. In such a region the continuous function $a \circ b$ takes a minimum and a maximum as well as all values in between. Therefore

$$\boldsymbol{a} \circ \boldsymbol{b} = [\min_{a \in \boldsymbol{a}, b \in \boldsymbol{b}} \{a \circ b\}, \max_{a \in \boldsymbol{a}, b \in \boldsymbol{b}} \{a \circ b\}] = [\inf(\boldsymbol{a} \circ \boldsymbol{b}), \sup(\boldsymbol{a} \circ \boldsymbol{b})],$$

i.e., for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{IR}$, $0 \notin \boldsymbol{b}$, $\boldsymbol{a} \circ \boldsymbol{b}$ is again an interval of $\mathbb{IR}$.

[6] An integral number a in an interval expression is interpreted as interval $[a, a]$.

Actually the minimum and maximum is taken for operations with the bounds. For bounded intervals $\boldsymbol{a} = [a_1, a_2]$ and $\boldsymbol{b} = [b_1, b_2]$ the following formula holds for all operations with $0 \notin \boldsymbol{b}$ in case of division:

$$\boldsymbol{a} \circ \boldsymbol{b} = \left[\min_{i,j=1,2}(a_i \circ b_j), \max_{i,j=1,2}(a_i \circ b_j) \right] \text{ for } \circ \in \{+, -, *, /\}. \tag{4}$$

We demonstrate this in case of addition. By (OD1) we obtain $a_1 \leq \boldsymbol{a}$ and $b_1 \leq \boldsymbol{b} \Rightarrow a_1 + b_1 \leq \inf(\boldsymbol{a} + \boldsymbol{b})$. On the other hand $\inf(\boldsymbol{a} + \boldsymbol{b}) \leq a_1 + b_1$. From both inequalities we obtain by (O3): $\inf(\boldsymbol{a} + \boldsymbol{b}) = a_1 + b_1$. Analogously one obtains $\sup(\boldsymbol{a} + \boldsymbol{b}) = a_2 + b_2$. Thus

$$\boldsymbol{a}+\boldsymbol{b} = [\min_{a\in\boldsymbol{a},b\in\boldsymbol{b}}\{a+b\}, \max_{a\in\boldsymbol{a},b\in\boldsymbol{b}}\{a+b\}] = [\inf(\boldsymbol{a}+\boldsymbol{b}), \sup(\boldsymbol{a}+\boldsymbol{b})] = [a_1 + b_1, a_2 + b_2].$$

Similarly by making use of (OD1,2,3,4) for intervals of $\mathbb{IR}$ and the simple sign rules $-(\boldsymbol{a} * \boldsymbol{b}) = (-\boldsymbol{a}) * \boldsymbol{b} = \boldsymbol{a} * (-\boldsymbol{b}), -(\boldsymbol{a}/\boldsymbol{b}) = (-\boldsymbol{a})/\boldsymbol{b} = \boldsymbol{a}/(-\boldsymbol{b})$ explicit formulas for all interval operations can be derived.

Now we get by (RG) for intervals of $\mathbb{IF}$

$$\boldsymbol{a} \diamondsuit \boldsymbol{b} := \diamondsuit(\boldsymbol{a} \circ \boldsymbol{b}) = \left[\triangledown \min_{i,j=1,2}(a_i \circ b_j), \triangle \max_{i,j=1,2}(a_i \circ b_j) \right]$$

and by the monotonicity of the roundings $\triangledown$ and $\triangle$:

$$\boldsymbol{a} \diamondsuit \boldsymbol{b} = \left[\min_{i,j=1,2}(a_i \mathbin{\triangledown} b_j), \max_{i,j=1,2}(a_i \mathbin{\triangle} b_j) \right].$$

For bounded and nonempty intervals $\boldsymbol{a} = [a_1, a_2]$ and $\boldsymbol{b} = [b_1, b_2]$ of $\mathbb{IF}$ the unary operation $-\boldsymbol{a}$ and the binary operations addition, subtraction, multiplication, and division are shown in the following tables. There the operator symbols for intervals are simply denoted by $+, -, *$, and $/$.

Minus operator $\quad -\boldsymbol{a} = [-a_2, -a_1].$

Addition $\quad [a_1, a_2] + [b_1, b_2] = [a_1 \mathbin{\triangledown} b_1, a_2 \mathbin{\triangle} b_2].$

Subtraction $\quad [a_1, a_2] - [b_1, b_2] = [a_1 \mathbin{\triangledown} b_2, a_2 \mathbin{\triangle} b_1].$

Multiplication $[a_1, a_2] * [b_1, b_2]$	$[b_1, b_2]$ $b_2 \leq 0$	$[b_1, b_2]$ $b_1 < 0 < b_2$	$[b_1, b_2]$ $b_1 \geq 0$
$[a_1, a_2], a_2 \leq 0$	$[a_2 \mathbin{\triangledown} b_2, a_1 \mathbin{\triangle} b_1]$	$[a_1 \mathbin{\triangledown} b_2, a_1 \mathbin{\triangle} b_1]$	$[a_1 \mathbin{\triangledown} b_2, a_2 \mathbin{\triangle} b_1]$
$a_1 < 0 < a_2$	$[a_2 \mathbin{\triangledown} b_1, a_1 \mathbin{\triangle} b_1]$	$[min(a_1 \mathbin{\triangledown} b_2, a_2 \mathbin{\triangledown} b_1),$ $max(a_1 \mathbin{\triangle} b_1, a_2 \mathbin{\triangle} b_2)]$	$[a_1 \mathbin{\triangledown} b_2, a_2 \mathbin{\triangle} b_2]$
$[a_1, a_2], a_1 \geq 0$	$[a_2 \mathbin{\triangledown} b_1, a_1 \mathbin{\triangle} b_2]$	$[a_2 \mathbin{\triangledown} b_1, a_2 \mathbin{\triangle} b_2]$	$[a_1 \mathbin{\triangledown} b_1, a_2 \mathbin{\triangle} b_2]$

Division, $0 \notin \boldsymbol{b}$	$[b_1, b_2]$	$[b_1, b_2]$
$[a_1, a_2]/[b_1, b_2]$	$b_2 < 0$	$b_1 > 0$
$[a_1, a_2], a_2 \leq 0$	$[a_2 \bigtriangledown b_1, a_1 \bigtriangleup b_2]$	$[a_1 \bigtriangledown b_1, a_2 \bigtriangleup b_2]$
$[a_1, a_2], a_1 < 0 < a_2$	$[a_2 \bigtriangledown b_2, a_1 \bigtriangleup b_2]$	$[a_1 \bigtriangledown b_1, a_2 \bigtriangleup b_1]$
$[a_1, a_2], 0 \leq a_1$	$[a_2 \bigtriangledown b_2, a_1 \bigtriangleup b_1]$	$[a_1 \bigtriangledown b_2, a_2 \bigtriangleup b_1]$

In real analysis division by zero is not defined. In interval arithmetic, however, the interval in the denominator of a quotient may contain zero. So this case has to be considered also.

The general rule for computing the set $\boldsymbol{a}/\boldsymbol{b}$ with $0 \in \boldsymbol{b}$ is to remove its zero from the interval $\boldsymbol{b}$ and perform the division with the remaining set.[7] Whenever zero in $\boldsymbol{b}$ is an endpoint of $\boldsymbol{b}$, the result of the division can be obtained directly from the above table for division with $0 \notin \boldsymbol{b}$ by the limit process $b_1 \to 0$ or $b_2 \to 0$ respectively. The results are shown in the following table. Here, the parentheses stress that the bounds $-\infty$ and $+\infty$ are not elements of the interval. When zero

Division, $0 \in \boldsymbol{b}$	$\boldsymbol{b} =$	$[b_1, b_2]$	$[b_1, b_2]$
$[a_1, a_2]/[b_1, b_2]$	$[0, 0]$	$b_1 < b_2 = 0$	$0 = b_1 < b_2$
$[a_1, a_2] = [0, 0]$	$\varnothing$	$[0, 0]$	$[0, 0]$
$[a_1, a_2], a_1 < 0, a_2 \leq 0$	$\varnothing$	$[a_2 \bigtriangledown b_1, +\infty)$	$(-\infty, a_2 \bigtriangleup b_2]$
$[a_1, a_2], a_1 < 0 < a_2$	$\varnothing$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[a_1, a_2], 0 \leq a_1, 0 < a_2$	$\varnothing$	$(-\infty, a_1 \bigtriangleup b_1]$	$[a_1 \bigtriangledown b_2, +\infty)$

is an interior point of the denominator, the set $[b_1, b_2]$ splits into the distinct sets $[b_1, 0]$ and $[0, b_2]$, and the division by $[b_1, b_2]$ actually means two divisions. The results of the two divisions are already shown in the table for division by $0 \in \boldsymbol{b}$.

However, in the user's program the two divisions appear as a single operation, as division by an interval $[b_1, b_2]$ with $b_1 < 0 < b_2$, an operation that delivers two distinct results.

A solution to the problem would be for the computer to provide a flag for *distinct intervals*. The situation occurs if the divisor is an interval that contains zero as an interior point. In this case the flag would be raised and signaled to the user. The user may then apply a routine of his choice to deal with the situation as is appropriate for his application. This routine could be: return the entire set of real numbers $(-\infty, +\infty)$ as result and continue the computation, or set a flag and continue the computation with one of the sets and ignore the other one, or put one of the sets on a list and continue the computation with the other one, or modify the operands and recompute, or stop computing, or some other action.

[7] This is in full accordance with function evaluation: When evaluating a function over a set, points outside its domain are simply ignored.

An alternative would be to provide a second division which in case of division by an interval that contains zero as an interior point generally delivers the result $(-\infty, +\infty)$. Then the user can decide when to use which division in his program.

In the operations defined so far it was assumed that the operands $\boldsymbol{a}$ and $\boldsymbol{b}$ are nonempty and bounded. Four kinds of extended intervals come from division by an interval of $\mathbb{IF}$ that contains zero:

$$\varnothing, \quad (-\infty, a], \quad [b, +\infty), \quad \text{and} \quad (-\infty, +\infty).$$

To extend the operations to these more general intervals the first rule is that any operation with the empty set $\varnothing$ returns the empty set. Then the above tables extend to possibly unbounded intervals of $\overline{\mathbb{IF}}$ by using the standard formulae for arithmetic operations involving $\pm\infty$ together with one rule that goes beyond standard rules for $\pm\infty$:

$$0 * (-\infty) = (-\infty) * 0 = 0 * (+\infty) = (+\infty) * 0 = 0.$$

This rule is not a new mathematical law, it is merely a short cut to compute the bounds of the result of multiplication on unbounded intervals.[8]

4 Interval Arithmetic in Higher Dimensional Spaces

The axioms for computer arithmetic shown in section 2 also can be applied to define computer arithmetic in higher dimensional spaces like complex numbers, vectors and matrices for real, complex, and interval data.

If M in section 2 is the set of real matrices and $S \subseteq M$ the set of floating-point matrices, then the definition of arithmetic operations in S by (RG) ideally requires exact evaluation of scalar products of vectors with floating-point components. Indeed very effective algorithms have been developed for computing scalar products of floating-point vectors exactly, [6].

If M in section 2 is the set of intervals of real vectors or matrices, respectively, then the set definition of arithmetic operations in M by (2) does not lead to an interval again. The result $\boldsymbol{a} \circ \boldsymbol{b} := \{a \circ b \mid a \in \boldsymbol{a} \wedge b \in \boldsymbol{b}\}$ is a more general set. It is an element of the power set[9] of vectors or matrices, respectively. To obtain an interval the upwardly directed rounding $\square$ from the power set onto the set of intervals of M has to be applied. With it arithmetic operations for intervals of M are defined by

$$\text{(RG)} \quad \boldsymbol{a} \boxdot \boldsymbol{b} := \square(\boldsymbol{a} \circ \boldsymbol{b}), \circ \in \{+, -, \cdots\}.$$

The set S of intervals of floating-point vectors or matrices, respectively, is a screen of M. To obtain arithmetic for intervals of S once more the monotone upwardly directed rounding, now denoted by $\diamondsuit$ is applied:

$$\text{(RG)} \quad \boldsymbol{a} \diamondsuit \boldsymbol{b} := \diamondsuit(\boldsymbol{a} \boxdot \boldsymbol{b}), \circ \in \{+, -, \cdots\}.$$

[8] Intervals of $\overline{\mathbb{IF}}$ are closed and connected sets of real numbers. So multiplication of any such interval by 0 can only have 0 as the result.

[9] The power set of a set M is the set of all subsets of M.

This leads to the best possible operations in the interval spaces M and S. So for intervals in higher dimensional spaces generally an additional rounding step is needed. It can be shown that in all relevant cases these best possible operations can be expressed by executable formulas just using the bounds of the intervals. Showing this for more complicated applications (e.g. complex interval matrices) requires development of a more general theory of computer arithmetic and more extensive studies of the applications. For details see [6] and the literature listed there.

5 Appendix

5.1 Hardware for Interval Arithmetic

The following figure gives a brief sketch of what hardware support for interval arithmetic may look like. It would not be hard to realize it in modern technology.

The circuitry broadly speaks for itself. The interval operands are loaded in parallel from a register file or a memory access unit. Then, after multiplexers have selected the appropriate operands, the lower bound of the result is computed with rounding downwards and the upper bound with rounding upwards with the selected operands. If in case of multiplication both operands contain zero as an interior point a second multiplication is necessary. In the figure it is assumed that the two multiplications are performed sequentially and that the results are delivered to the lower part of the circuitry. They are then forwarded to a comparison unit. Here for the lower bound of the result the lower and for the upper bound the higher of the two products is selected. This lower part of the circuitry could also be used to perform comparison relations.

Table 1 shows the control signals for the operand selection by the multiplexers. These signals are computed from the signs of the bounds of the interval operands $\boldsymbol{a} = [a_1, a_2]$ and $\boldsymbol{b} = [b_1, b_2]$. A negative sign is a 1. A bar upon a logical value means inversion. In the expressions a dot stands for a logical and, and a plus for a logical or. In case of multiplication the signal ms is zero if only one product pair is to be computed, and it is one if a second product pair is to be computed.

Every operand selector signal can be realized by two or three gates! For more details see [3] or [6].

Table 1. Operand selection signals

OS	o_{a1}	o_{a2}	o_{b1}	o_{b2}
$+$	0	1	0	1
$-$	0	1	1	0
$*$	$s_{b2} + \overline{s_{a1}} \cdot s_{b1} + ms\ \overline{s_{b1}} + \overline{s_{a1}} \cdot \overline{s_{b2}} + ms$		$\overline{ms}(s_{a2} + s_{a1} \cdot \overline{s_{b2}})$	$\overline{s_{a1}} + \overline{s_{a2}} \cdot \overline{s_{b1}} + ms$
$/$	$s_{b2} + s_{a1} \cdot s_{b1}$	$\overline{s_{b1}} + s_{a2} \cdot \overline{s_{b2}}$	$\overline{s_{a1}} + \overline{s_{a2}} \cdot s_{b1}$	$s_{a2} + s_{a1} \cdot s_{b1}$

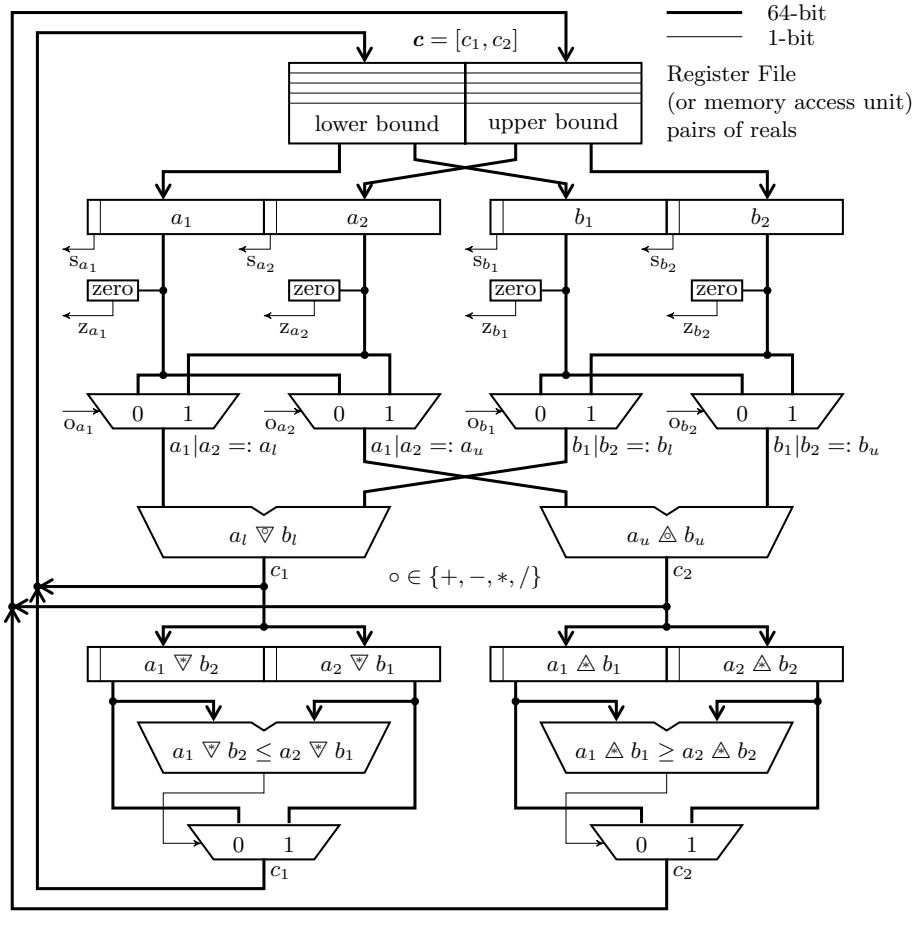


Fig. 1. Circuitry for Interval Operations

5.2 Interval Arithmetic on X86-Processors

It is interesting to note that most of what is needed for fast hardware support for interval arithmetic is already available on current x86-processors.

On an Intel Pentium 4, for instance, eight registers are available for words of 128 bits (xmm0, xmm1, ..., xmm7). The x86-64 processors even provide 16 such registers. These registers can hold pairs of double precision floating-point numbers. They can be viewed as bounds of intervals. Parallel operations like +, −, ·, /, min, max, and compare can be performed on these pairs of numbers. What is not available and would be needed is for one of the two operations to be rounded downwards and the other one rounded upwards. Even shuffling of

bounds is possible under certain conditions. This is half of operand selection needed for interval arithmetic. So an interval operation would need two such units or to pass this unit twice. Also nearly all of the data paths are available on current x86-processors. Thus full hardware support of interval arithmetic would probably add very very little to a current Intel or AMD x86 processor chip.

Full hardware support of fast interval arithmetic on RISC processors may cost a little more as these lack pairwise processing. But most of them have two arithmetic units and use them for super scalar processing. What has to be added is some sophisticated control.

References

1. American National Standards Institute / Institute of Electrical and Electronics Engineers: A Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-1987, New York (1985) (reprinted in SIGPLAN 22(2), 9–25, 1987); Also adopted as IEC Standard 559:1989
2. American National Standards Institute / Institute of Electrical and Electronics Engineers: A Standard for Radix-Independent Floating-Point Arithmetic. ANSI/IEEE Std. 854-1987, New York (1987)
3. Kirchner, R., Kulisch, U.: Hardware support for interval arithmetic. Reliable Computing 12(3), 225–237 (2006)
4. Kulisch, U.: Implementation and Formalization of Floating-Point Arithmetics. IBM T. J. Watson-Research Center, Report Nr. RC 4608, 1–50 (1973); Invited talk at the Caratheodory Symposium, September 1973 in Athens, published in: The Greek Mathematical Society, C. Caratheodory Symposium, 328–369 (1973), and in Computing 14, 323–348 (1975)
5. Kulisch, U.: Grundlagen des Numerischen Rechnens - Mathematische Begründung der Rechnerarithmetik, Bibliographisches Institut, Mannheim Wien Zürich (1976)
6. Kulisch, U.: Computer Arithmetic and Validity – Theory, Implementation, and Applications. de Gruyter, Berlin (2008)
7. Kulisch, U.: Arithmetic Operations for Floating-Point Intervals, as Motion 5 accepted by the IEEE Standards Committee P1788 as definition of the interval operations [8]
8. Pryce, J.D. (ed.): P1788, IEEE Standard for Interval Arithmetic, http://grouper.ieee.org/groups/1788/email/pdfOWdtH2mOd9.pdf

A Method for Comparing Intervals with Interval Bounds

Pavel Sevastjanov, Pavel Bartosiewicz, and Kamil Tkacz

Institute of Computer & Information Sciences
Czestochowa University of Technology
Dabrowskiego 73, 42-200 Czestochowa, Poland
sevast@icis.pcz.pl

Abstract. There are different methods for interval comparison used in modeling, optimization and decision making in interval setting. These methods make it possible to compare intervals with real valued bounds. Nevertheless, in practice, for example in the rule-base evidential reasoning in interval setting, the problem of comparing intervals with interval bounds arises. In this report, a method for comparison of intervals with interval bounds is proposed and illustrated using numerical examples. This method is based on the mathematical tools of the Dempster-Shafer theory of evidence.

Keywords: Intervals with interval bounds, Interval comparison, Dempster-Shafer theory of evidence.

1 Introduction

The problem of interval comparison is of perennial interest, because of its direct relevance in practical modeling and optimization of real world processes. Moreover, sometimes in practice we meet situations when intervals with interval bounds must be compared. For example, the developed in [3] trading system based on the synthesis of Fuzzy Sets theory (FST) and the Dempster-Shafer theory (DST) provides the results in the form of belief intervals which must be compared to generate buying or selling signals. It is easy to show that if input data in this system are presented by intervals, the final estimates of buying or selling signals will be belief intervals with interval bounds, which should be compared to select final buying or selling signals.

Therefore, in this report we propose a method for comparing intervals with interval bounds based on the method for the comparison of usual interval with real valued bounds. To compare intervals with the real valued bounds, usually the quantitative indices are used (see reviews in [5] and [7]).

In the current report, we use the DST formalism to get the results of interval comparison in the interval form. The reason behind this is that all arithmetic operations on intervals provide interval results. Therefore, it seems quite natural to expect an interval result of interval comparison too.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 496–503, 2012.

Here we present some basic definitions of DST needed for the subsequent analysis.

The origins of the Dempster-Shafer theory go back to the work by A.P. Dempster [1,2] who developed a system of upper and lower probabilities. Following this work his student G. Shafer [6] included in his 1976 book "A Mathematical Theory of Evidence" a more thorough explanation of belief functions.

Assume A are subsets of X. It is important to note that a subset A may be treated also as a question or proposition and X as a set of propositions or mutually exclusive hypotheses or answers. A DS belief structure has associated with it a mapping m, called basic assignment function, from subsets of X into a unit interval, $m : 2^X \to [0, 1]$ such that $m(\emptyset) = 0$, $\sum_{A \subseteq X} m(A) = 1$. The subsets of X for which the mapping does not assume a zero value are called focal elements.

In the framework of classical Dempster-Shafer approach, it is assumed that the null set is never a focal element.

In [6], Shafer introduced a number of measures associated with DS belief structure. The measure of belief is a mapping $Bel : 2^X \to [0, 1]$ such that for any subset B of X

$$Bel(B) = \sum_{\emptyset \neq A \subseteq B} m(A). \tag{1}$$

It is shown in [6] that m can be uniquely recovered from Bel. A second measure introduced by Shafer [6] is a measure of plausibility. The measure of plausibility associated with m is a mapping $Pl : 2^X \to [0, 1]$ such that for any subset B of X.

$$Pl(B) = \sum_{A \cap B \neq \emptyset} m(A). \tag{2}$$

It is easy to see that $Bel(B) \leq Pl(B)$. DS provides an explicit measure of ignorance about an event B and its complementary $\overline{B}$ as a length of an interval $[Bel(B), Pl(B)]$ called the belief interval (BI). It can also be interpreted as imprecision of the "true probability" of B [6].

The rest of the paper is set out as follows. In Section 2, we describe the method for interval comparison using DST proposed in [5]. Section 3 presents a method for comparison of intervals with interval bounds. Section 4 concludes with some remarks.

2 Interval Comparison Using *DST*

Since we have presented the method for interval comparison with the use of DST earlier in [5] in detail, in this section only its brief description is performed. There are only two non-trivial cases of interval locations which we call overlapping and inclusion cases (see Fig.1) deserve to be considered. Let $A = [a_1, a_2]$ and $B = [b_1, b_2]$ be independent intervals and $a \in [a_1, a_2]$, $b \in [b_1, b_2]$ be random values distributed on these intervals. As we are dealing with usual intervals, the natural assumption is that the random values a and b are distributed uniformly. There are some

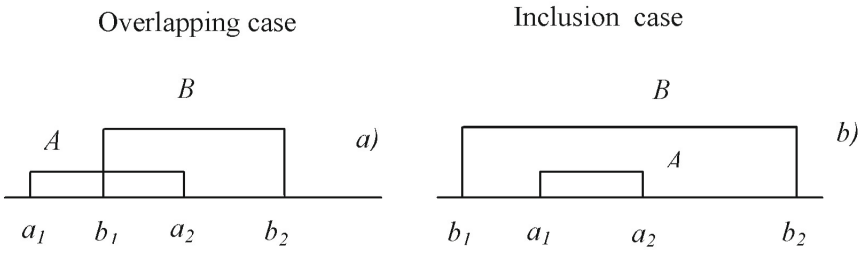


Fig. 1. The examples of interval relations

subintervals which play an important role in our analysis. For example (see Fig. 1a), the falling of random $a \in [b_1, b_2]$, $b \in [a_1, a_2]$ into subintervals $[a_1, b_1]$, $[b_1, a_2]$, $[a_2, b_2]$ may be treated as a set of independent random events.

Let us consider the case of overlapping intervals (Fig.1a). Only four mutually exclusive events may take place in considered situation: $H_1 : a \in [a_1, b_1] \& b \in [a_2, b_2]$, $H_2 : a \in [a_1, b_1] \& b \in [b_1, a_2]$, $H_3 : a \in [b_1, a_2] \& b \in [b_1, a_2]$, $H_4 : a \in [b_1, a_2] \& b \in [a_2, b_2]$. For the probabilities of events H_1–H_4 from the simple geometric reasons we obtain

$$P(H_1) = \frac{b_1 - a_1}{a_2 - a_1}\frac{b_2 - a_2}{b_2 - b_1}, \qquad P(H_2) = \frac{b_1 - a_1}{a_2 - a_1}\frac{a_2 - b_1}{b_2 - b_1},$$
$$P(H_3) = \frac{a_2 - b_1}{a_2 - a_1}\frac{a_2 - b_1}{b_2 - b_1}, \qquad P(H_4) = \frac{a_2 - b_1}{a_2 - a_1}\frac{b_2 - a_2}{b_2 - b_1}. \tag{3}$$

It can easily be proved that

$$P(H_1) + P(H_2) + P(H_3) + P(H_4) = 1. \tag{4}$$

Thus, in the sense of DST, the probabilities $P(H_i)$, $i = 1$ to 4, can be used to construct a basic assignment function m. It has been shown in [5] that in the case of overlapping intervals ($a_1 < b_1$ and $a_2 < b_2$), there are only two interval relations which make a sense: $A < B$ and $A = B$. It is easy to see that events H_1, H_2 and H_4 may be considered as the "strong" evidences of $A < B$, otherwise H_3 can be treated as only the "weak" evidence of $A < B$ because it simultaneously is the witness of $A = B$. In the DST notation, we obtain:

$$m(A < B) = P(H_1) + P(H_2) + P(H_4), m(A < B, A = B) = P(H_3). \tag{5}$$

Then from (3), (4) and (5) we get

$$Bel(A < B) = m(A < B) = 1 - P(H_3) = 1 - \frac{(a_2 - b_1)^2}{(a_2 - a_1)(b_2 - b_1)}, \tag{6}$$

$$Pl(A < B) = m(A < B) + m(A < B, A = B) = 1. \tag{7}$$

In the similar way, the pair of estimations for $A = B$ has been inferred:

$$Bel(A = B) = 0, \quad Pl(A = B) = P(H_3) = \frac{(a_2 - b_1)^2}{(a_2 - a_1)(b_2 - b_1)}. \tag{8}$$

Then belief intervals BI may be presented as follows:

$$BI(A < B) = [Bel(A < B), Pl(A < B)] = [Bel(A < B, 1]. \tag{9}$$

So using an approach based on DST, we obtain the interval estimations for the degree of interval inequality and equality. It is worth noting that introduced interval form of interval comparison estimations is a real embodiment of usually implicitly expressed, but pivotal requirement of interval arithmetic: the result of interval operation should be an interval too.

Let us consider the inclusion case (Fig. 1b). In this case we have three possible events: $H_1 : a \in [a_1, a_2] \& b \in [b_1, a_1]$, $H_2 : a \in [a_1, a_2] \& b \in [a_1, a_2]$, $H_3 : a \in [a_1, a_2] \& b \in [a_2, b_2]$.

Since $b_1 \leq a_1$, in this case the relation $A > B$ may become true. For instance, there are no doubts that $A > B$ if $b_1 < a_1$ and $b_2 = a_2$. We can observe the elementary evidences of events $A < B$, $A = B$, $A > B$ in this situation and we can take them into account to construct the Bel and Pl functions using nearly the same reasoning as in the case of overlapping intervals. Finally, we get

$$BI(A < B) = [Bel(A < B), Pl(A < B)] = \left[\frac{b_2 - a_2}{b_2 - b_1}, \frac{b_2 - a_1}{b_2 - b_1}\right], \tag{10}$$

$$BI(A = B) = [Bel(A = B), Pl(A = B)] = \left[0, \frac{a_2 - a_1}{b_2 - b_1}\right], \tag{11}$$

$$BI(A > B) = [Bel(A > B), Pl(A > B)] = \left[\frac{a_1 - b_1}{b_2 - b_1}, \frac{a_2 - b_1}{b_2 - b_1}\right]. \tag{12}$$

It is shown in [5] that in both cases (overlapping and inclusion) there is always an intersection of the belief intervals representing equality and inequality relations. Thus, the structure of obtained results makes it possible to use the simplest method for the belief intervals comparison similar to the approach by Moore [4]. In a nutshell, if A and B are belief intervals, we say $A < B$ if $a_1 \leq b_1$, $a_2 \leq b_2$ and at least one of the last two inequalities is strong.

Indeed, several real valued criteria may be applied in order to make a reasonable final choice when comparing intervals basing on the degrees to which one interval is greater/smaller than other [5].

3 The Use of *DST* for Comparison of Intervals with Interval Bounds

Here we use the described above method for interval comparison to develop a method for the comparison of intervals with interval bounds.

Let us consider such interval $[B] = \left[[b^L], [b^U]\right]$, where $[b^L] = [\underline{b}^L, \overline{b}^L]$ and $[b^U] = [\underline{b}^U, \overline{b}^U]$. This interval $[B]$ can be treated as a continuous set $\{[x]\}$ of usual intervals $[x] = [x^L, x^U]$ such that $x^L \in [\underline{b}^L, \overline{b}^L]$ and $x^U \in [\underline{b}^U, \overline{b}^U]$.

Since it is possible that $\overline{b}^L \geq \underline{b}^U$ (see Fig. 2), it is important to note that this set of intervals should comprise only regular intervals, i.e., such that $x^L \leq x^U$.

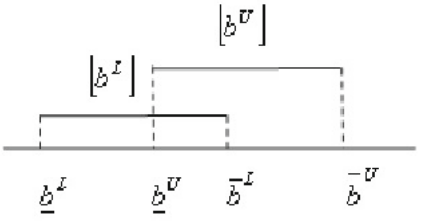


Fig. 2. Interval with overlapping interval bounds

Similarly, the interval $[A] = \left[[\underline{a}^L, \bar{a}^L], [\underline{a}^U, \bar{a}^U]\right]$ can be presented by the set $\{[y]\}$ of regular intervals $[y] = [y^L, y^U]$ such that $y^L \in [\underline{a}^L, \overline{a}^L]$ and $y^U \in [\underline{a}^U, \overline{a}^U]$.

It seems obvious that generally if we have to compare the intervals $[B]$ and $[A]$, all intervals from the set $\{[x]\}$ should be compared with those from the set $\{[y]\}$.

On the other hand, it is clear that this problem can be simplified, as to compare $[B]$ and $[A]$ it is enough to compare only the intervals presented by the bounds of interval bounds of intervals $[B]$ and $[A]$.

For example, for the interval $[A] = \left[[\underline{a}^L, \bar{a}^L], [\underline{a}^U, \bar{a}^U]\right]$ we have four such intervals: $[\underline{a}^L, \underline{a}^U]$, $[\underline{a}^L, \bar{a}^U]$, $[\bar{a}^L, \underline{a}^U]$, $[\bar{a}^L, \bar{a}^U]$ if $\bar{a}^L \leq \underline{a}^U$ or $[\underline{a}^L, \underline{a}^U]$, $[\underline{a}^L, \bar{a}^U]$, $[\underline{a}^U, \bar{a}^L]$, $[\bar{a}^L, \bar{a}^U]$ if $\bar{a}^L > \underline{a}^U$. We shall denote these intervals as $[a]_i$, $i = 1, 2, 3, 4$.

Obviously, for the interval $[B] = \left[[b^L], [b^U]\right]$ we have four similar intervals $[b]_i$, $i = 1, 2, 3, 4$, too.

Therefore, to compare intervals $[B]$ and $[A]$ we should compare each of four defined above intervals belonging to $[B]$ with each of four similar intervals belonging to $[A]$. Then using the described above method for comparison of usual intervals, we obtain sixteen estimates in the form of belief intervals: $BI([b]_i > [a]_j)$, $BI([b]_i < [a]_j)$, $BI([b]_i = [a]_j)$, $i, j = 1, 2, 3, 4$.

To get the final estimates, we propose to use the following averaging:

$$BI([B] > [A]) = \frac{1}{16} \sum_{i=1}^{4} \sum_{j=1}^{4} BI([b]_i > [a]_j), \tag{13}$$

$$BI([B] < [A]) = \frac{1}{16} \sum_{i=1}^{4} \sum_{j=1}^{4} BI([b]_i < [a]_j), \tag{14}$$

$$BI([B] = [A]) = \frac{1}{16} \sum_{i=1}^{4} \sum_{j=1}^{4} BI([b]_i = [a]_j). \tag{15}$$

Obviously, the resulting $BI([B] > [A])$, $BI([B] < [A])$ and $BI([B] = [A])$ are usual belief intervals. Therefore, to compare them, the method described in the previous section can be applied.

The proposed method is based on the averaging and can be treated as a heuristic approach. Therefore, let us consider some numerical example which

confirm its validity. In our analysis, we shall use the intuitively clear Moore's assumption [4]: if $A = [\underline{a}, \overline{a}]$ and $B = [\underline{b}, \overline{b}]$ are regular intervals, we say $A < B$ if $\underline{a} \leq \underline{b}$ and $\overline{a} \leq \overline{b}$ and at least one of the last two inequalities is strong.

Example 1
Let us consider the following intervals with interval bounds:

$[A] = [[0, 3], [5, 8]]$, $[B] = [[1, 3], [5, 8]]$ and $[C] = [[2, 3], [5, 8]]$ (see Fig.3).

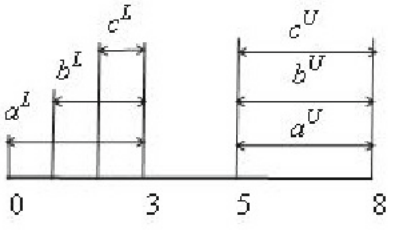


Fig. 3. Interval with overlapping interval bounds. Example 1.

As the right interval bounds of these intervals are equal, $\underline{a}^U = \underline{b}^U = \underline{c}^U$ and $\underline{a}^L < \underline{b}^L < \underline{c}^L$ we can conclude that $[A] < [B] < [C]$.

Using our method we have obtained:

$BI([B] > [A]) = [0.263, 0.786]$, $BI([B] < [A]) = [0.201, 0.698]$,
$BI([B] = [A]) = [0, 0.535]$,
$BI([C] > [B]) = [0.258, 0.805]$, $BI([C] < [B]) = [0.179, 0.706]$,
$BI([C] = [B]) = [0, 0.563]$,
$BI([C] > [A]) = [0.306, 0.811]$, $BI([C] < [A]) = [0.172, 0.666]$,
$BI([C] = [A]) = [0, 0.523]$.

It is clear that according to the Moore's assumption

$BI([B] = [A]) < BI([B] < [A]) < BI([B] > [A])$ and therefore $[B] > [A]$,
$BI([C] = [B]) < BI([C] < [B]) < BI([C] > [B])$ and therefore $[C] > [B]$,
$BI([C] = [A]) < BI([C] < [A]) < BI([C] > [A])$ and therefore $[C] > [A]$.

Since these results lead to $[A] < [B] < [C]$, we can say that our method provides the true result.

Example 2
Let us compare the intervals:

$[A] = [[0, 3], [5, 8]]$, $[B] = [[0, 3], [4, 8]]$, $[C] = [[0, 3], [3, 8]]$ (see Fig.4).

Since the left interval bounds of these intervals are equal, $\overline{c}^U = \overline{b}^U = \overline{a}^U$ and $\underline{c}^U < \underline{b}^U < \underline{a}^U$, we can conclude that $[C] < [B] < [A]$.

Using our method we have obtained

$BI([B] > [A]) = [0.221, 0.659]$, $BI([B] < [A]) = [0.330, 0.769]$,
$BI([B] = [A]) = [0, 0.449]$,

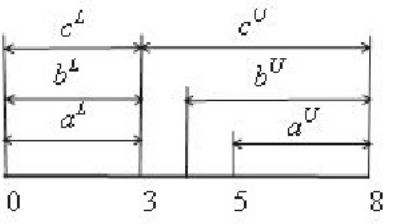


Fig. 4. Interval with overlapping interval bounds. Example 2.

$BI([C] > [B]) = [0.266, 0.596]$, $BI([C] < [B]) = [0.406, 0.731]$,
$BI([C] = [B]) = [0, 0.328]$,
$BI([C] > [A]) = [0.221, 0.575]$, $BI([C] < [A]) = [0.425, 0.769]$,
$BI([C] = [A]) = [0, 0.354]$.

According to the Moore's assumption we have:

$BI([B] = [A]) < BI([B] > [A]) < BI([B] < [A])$ and therefore $[B] < [A]$,
$BI([C] = [B]) < BI([C] > [B]) < BI([C] < [B])$ and therefore $[C] < [B]$,
$BI([C] = [A]) < BI([C] > [A]) < BI([C] < [A])$ and therefore $[C] < [A]$.

We can see that our method gives the true result: $[C] < [B] < [A]$.

Example 3
In this case, we deal with the type of intervals shown in Fig. 5.:

$[A] = [[0, 5], [4, 8]]$, $[B] = [[0, 5], [3, 8]]$, $[C] = [[0, 5], [2, 8]]$ (see Fig.5).

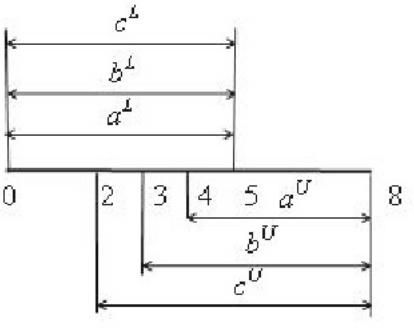


Fig. 5. Interval with overlapping interval bounds. Example 3.

The left interval bounds of these intervals are equal, $\overline{c}^U = \overline{b}^U = \overline{a}^U$ and $\underline{c}^U < \underline{b}^U < \underline{a}^U$. Then we can conclude that $[C] < [B] < [A]$.

Using our method we get:

$BI([B] > [A]) = [0.297, 0.633]$, $BI([B] < [A]) = [0.367, 0.695]$,
$BI([B] = [A]) = [0, 0.336]$,
$BI([C] > [B]) = [0.298, 0.638]$, $BI([C] < [B]) = [0.362, 0.695]$,
$BI([C] = [B]) = [0, 0.340]$,
$BI([C] > [A]) = [0.276, 0.599]$, $BI([C] < [A]) = [0.401, 0.703]$,
$BI([C] = [A]) = [0, 0.323]$.

According to the Moore's assumption we have:

$BI([B] = [A]) < BI([B] > [A]) < BI([B] < [A])$ and therefore $[B] < [A]$,
$BI([C] = [B]) < BI([C] > [B]) < BI([C] < [B])$ and therefore $[C] < [B]$,
$BI([C] = [A]) < BI([C] > [A]) < BI([C] < [A])$ and therefore $[C] < [A]$.

We can see that our method gives the true result: $[C] < [B] < [A]$.

These three examples make it possible to say that the developed method for comparison of intervals with interval bounds is reliable enough and provides true results.

4 Conclusion

The mathematical tools of the Dempster-Shafer theory of evidence are used to develop a method for usual interval comparison which provide the results of comparison in the form of belief intervals. This method is used as the base for developing the method for comparison of intervals with interval bounds. The presented numerical examples make it possible to say that the developed method for comparison of intervals with interval bounds is reliable enough and provides true intuitively obvious results.

References

1. Dempster, A.P.: Upper and lower probabilities induced by a muilti-valued mapping. Ann. Math. Stat. 38, 325–339 (1967)
2. Dempster, A.P.: A generalization of Bayesian inference (with discussion). J. Roy. Stat. Soc., Series B 30, 208–247 (1968)
3. Dymova, L., Sevastianov, P., Bartosiewicz, P.: A new approach to the rule-base evidential reasoning: stock trading decision support system application. Expert Systems with Applications 37, 5464–5576 (2010)
4. Moore, R.E.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)
5. Sevastjanov, P.: Numerical methods for interval and fuzzy number comparison based on the probabilistic approach and Dempster-Shafer theory. Information Sciences 177, 4645–4661 (2007)
6. Shafer, G.: A mathematical theory of evidence. Princeton University Press, Princeton (1976)
7. Wang, X., Kerre, E.E.: Reasonable properties for the ordering of fuzzy quantities (I) (II). Fuzzy Sets and Systems 112, 387–405 (2001)

Direct Interval Extension of TOPSIS Method

Pavel Sevastjanov and Anna Tikhonenko

Institute of Comp. & Information Sci., Czestochowa University of Technology,
Dabrowskiego 73, 42-201 Czestochowa, Poland
sevast@icis.pcz.pl

Abstract. The technique for order preference by similarity to ideal solution ($TOPSIS$) currently is one of most popular methods for Multiple criteria decision making ($MCDM$).This technique was primary developed for dealing with only real-valued data. In some cases, determining precisely the exact values of local criteria is difficult and as a result their values are considered as intervals. There are several papers devoted to interval extensions of $TOPSIS$ in the literature, but these extensions are not completed as ideal solutions are presented by real values, not by intervals.

In this report, we show that these extensions may lead to the wrong results especially in the case of intersection of some intervals representing the values of criteria. Therefore, we propose a new direct approach to interval extension of $TOPSIS$ method which is free of limitations of known methods.

Keywords: Interval extension, TOPSIS method.

1 Introduction

The technique for order performance by similarity to ideal solution ($TOPSIS$) [5], is one of known classical $MCDM$ method. It was first developed by Hwang and Yoon [2] for solving a $MCDM$ problem.

The basic principle of the $TOPSIS$ method is that the chosen alternative should have the shortest distance from the positive ideal solution and the farthest distance from the negative ideal solution. There exist a large amount of literature involving $TOPSIS$ theory and applications. In classical $MCDM$ methods, the ratings and weights of criteria are known precisely. A survey of these methods has been presented in [2]. In the classical $TOPSIS$ method, the performance ratings and the weights of criteria are given exact values.

Nevertheless, sometimes determining precisely the exact values of criteria is difficult and as a result, their values are presented by intervals.

Jahanshahloo et al. [3,4], extended the concept of $TOPSIS$ method to develop a methodology for solving $MCDM$ problem with interval data. The main limitation of this approach is that the ideal solutions are presented by real values, not by intervals. The similar approach to determining ideal solutions is used in [13].

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 504–512, 2012.

In this report, we show that these extensions may lead to the wrong results especially in the case of intersection of some intervals representing the values of criteria.

Therefore, we propose a new direct approach to interval extension of $TOPSIS$ method which is free of the limitations of known approaches.

The rest of the paper is set out as follows. In Section 2, we present the basics of $TOPSIS$ method and its known interval extension. Section 3 presents the direct interval extension of $TOPSIS$ method. The results obtained using the method proposed in [3,4] are compared with those obtained by the proposed new method. Section 4 concludes with some remarks.

2 The Basics of *TOPSIS* Method and Known Approach to Its Interval Extension

The classical $TOPSIS$ method is based on the idea that the best alternative should have the shortest distance from the positive ideal solution and the farthest distance from the negative ideal solution. It is assumed that if each local criterion takes monotonically increasing or decreasing variation, then it is easy to define an ideal solution.

The positive ideal solution is composed of all the best achievable values of local criteria, while the negative ideal solution is composed of all the worst achievable values of local criteria.

Suppose a $MCDM$ problem is based on m alternatives A_1, A_2,...,A_m and n criteria C_1, C_2,...,C_n. Each alternative is evaluated with respect to the n criteria. All the ratings are assigned to alternatives with respect to decision matrix $D[x_{ij}]_{n\times m}$, where x_{ij} is the rating of alternative A_i with respect to the criterion C_j. Let $W = [w_1, w_2, ..., w_n]$ be the vector of local criteria weights satisfying $\sum_{j=1}^{n} w_j = 1$.

The $TOPSIS$ method consists of the following steps:

1. Normalize the decision matrix:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=1}^{m} x_{kj}^2}},\ i = 1, ..., m;\ j = 1, ..., n. \tag{1}$$

Multiply the columns of normalized decision matrix by the associated weights:

$$v_{ij} = w_j \times r_{ij},\ i = 1, ..., m;\ j = 1, ..., n. \tag{2}$$

2. Determine the positive ideal and negative ideal solutions, respectively, as follows::

$$\begin{aligned} &\mathrm{A}^+ = \{v_1^+, v_2^+, ..., v_n^+\} = \\ &= \{(\max_i v_{ij} \,|j \in K_b),\ (\min_i v_{ij} \,|j \in K_c)\}, \end{aligned} \tag{3}$$

$$\begin{aligned} &\mathrm{A}^- = \{v_1^-, v_2^-, ..., v_n^-\} = \\ &= \{(\min_i v_{ij} \,|j \in K_b),\ (\max_i v_{ij} \,|j \in K_c)\}, \end{aligned} \tag{4}$$

where K_b is the set of benefit criteria and K_c is the set of cost criteria.

3. Obtain the distances of the existing alternatives from positive ideal and negative ideal solutions: two Euclidean distances for each alternatives are, respectively, calculated as follows:

$$\begin{aligned} S_i^+ &= \sqrt{\sum_{j=1}^n (v_{ij} - v_j^+)^2},\ i = 1, ..., m, \\ S_i^- &= \sqrt{\sum_{j=1}^n (v_{ij} - v_j^-)^2},\ i = 1, ..., m. \end{aligned} \tag{5}$$

4. Calculate the relative closeness to the ideal alternatives:

$$RC_i = \frac{S_i^-}{S_i^+ + S_i^-},\ i = 1, 2, ..., m,\ 0 \le RC_i \le 1. \tag{6}$$

5. Rank the alternatives according to the relative closeness to the ideal alternatives: the bigger is the RC_i, the better is the alternative A_i.

In [3,4], an interval extension of classical $TOPSIS$ method is proposed. This approach may be described as follows.

Let $[x_{ij}^L, x_{ij}^U]$ be an interval value of jth criterion for ith alternative (x_{ij}^L and x_{ij}^U are the lower and upper bounds of interval, respectively), $W = [w_1, w_2, ..., w_n]$ be the weight vector satisfying $\sum_{j=1}^n w_j = 1$. Then $D[[x_{ij}^L, x_{ij}^U]]_{n \times m}$ is the interval valued decision matrix. The method proposed in [3,4] consists of the following steps:

1. Normalizing the decision matrix using the following expressions:

$$r_{ij}^L = \frac{x_{ij}^L}{\left(\sum\limits_{k=1}^m ((x_{kj}^L)^2 + (x_{kj}^U)^2) \right)^{\frac{1}{2}}},\ i = 1, ..., m;\ j = 1, ..., n, \tag{7}$$

$$r_{ij}^U = \frac{x_{ij}^U}{\left(\sum\limits_{k=1}^m ((x_{kj}^L)^2 + (x_{kj}^U)^2) \right)^{\frac{1}{2}}},\ i = 1, ..., m;\ j = 1, ..., n. \tag{8}$$

2. Taking into account the importance of criteria, the weighted normalized interval decision matrix is obtained using the following expressions:

$$v_{ij}^L = w_j \times r_{ij}^L,\ \ v_{ij}^U = w_j \times r_{ij}^U,\ i = 1, ..., m;\ j = 1, ..., n.$$

3. The positive and negative ideal solutions are obtained as follows:

$$\begin{aligned} &A^+ = \{v_1^+, v_2^+, ..., v_n^+\} = \\ &= \left\{ (\max_i v_{ij}^U \,|\, j \in K_b), (\min_i v_{ij}^L \,|\, j \in K_c \right\}, \end{aligned} \tag{9}$$

$$\begin{aligned} &A^- = \{v_1^-, v_2^-, ..., v_n^-\} = \\ &= \left\{ (\min_i v_{ij}^L \,|\, j \in K_b), (\max_i v_{ij}^U \,|\, j \in K_c \right\}. \end{aligned} \tag{10}$$

4. The separation of each alternative from the positive ideal solution is calculated using the n -dimensional Eucludean distance:

$$S_i^+ = \left\{ \sum_{j \in K_b} (v_{ij}^L - v_j^+)^2 + \sum_{j \in K_c} (v_{ij}^U - v_j^+)^2 \right\}^{\frac{1}{2}}, \; i = 1, ..., m. \tag{11}$$

Similarly, the separation from the negative ideal solution is calculated as follows:

$$S_i^- = \left\{ \sum_{j \in K_b} (v_{ij}^U - v_j^-)^2 + \sum_{j \in K_c} (v_{ij}^L - v_j^-)^2 \right\}^{\frac{1}{2}}, \; i = 1, ..., m. \tag{12}$$

5. Calculate the relative closeness to the ideal alternatives:

$$RC_i = \frac{S_i^-}{S_i^+ + S_i^-}, \; i = 1, 2, ..., m, \; 0 \leq RC_i \leq 1. \tag{13}$$

6. Rank the alternatives: according to the relative closeness to the ideal alternatives, the bigger is the RC_i: the better is the alternative A_i.

3 The Direct Interval Extension of *TOPSIS* Method

3.1 The Problem Formulation

We can see that in expressions (9) and (10) the maximal and minimal values achievable in intervals (the bounds of intervals $[v_{ij}^L, v_{ij}^U]$) are used instead of corresponding maximal and minimal interval values themselves. It is implicitly assumed that the bounds of intervals may represent their interval values. Nevertheless, such approach seems to be justified only in the case when there are no any intersections of these intervals. In the cases, when such intersection exist, this approach may lead to wrong results. For example, let us consider two intervals $[x_{11}]$=[5, 7] and $[x_{21}]$=[0, 10] which represent the ratings of alternatives A_1 and A_2 with respect to the benefit criterion C_1.

Then using expressions (9) and (10) we get $v_1^+ = 10$ and $v_1^- = 0$, whereas using any method for interval comparison (see below) we obtain that $[x_{11}] > [x_{21}]$ and for the positive and negative interval ideal solutions we get: $[v]_1^+$=[5,7] and $[v]_1^-$=[0,10], respectively. We can see that v_1^+=10 is not even included in $[v]_1^+$.

Therefore, a more correct approach to calculation of ideal solutions is representing them in the interval form using the expressions:

$$\begin{aligned} A^+ &= \{[v_1^{+L}, v_1^{+U}], [v_2^{+L}, v_2^{+U}], ..., [v_n^{+L}, v_n^{+U}]\} = \\ &= \left\{ (\max_i [v_{ij}^L, v_{ij}^U] \,|\, j \in K_b), \; (\min_i [v_{ij}^L, v_{ij}^U] \,|\, j \in K_c) \; \right\}, \end{aligned} \tag{14}$$

$$\begin{aligned} A^- &= \{[v_1^{-L}, v_1^{-U}], [v_2^{-L}, v_2^{-U}], ..., [v_n^{-L}, v_n^{-U}]\} = \\ &= \left\{ (\min_i [v_{ij}^L, v_{ij}^U] \,|\, j \in K_b), \; (\max_i [v_{ij}^L, v_{ij}^U] \,|\, j \in K_c) \; \right\}. \end{aligned} \tag{15}$$

As there are no any type reductions (representation of intervals by real values) in (14) and (15) we call our approach Direct Interval Extension of $TOPSIS$ method.

As in (14) and (15) the minimal and maximal intervals must be chosen, the main difficulty in the implementation of the above method is the problem of interval comparison.

3.2 Interval Comparison

The problem of interval comparison is of perennial interest, because of its direct relevance in practical modeling and optimization of real-world processes.

To compare intervals, usually the quantitative indices are used (see reviews in [8] and [7]). Wang at al. [9] proposed a simple heuristic method which provides the degree of possibility that an interval is greater/lesser than another one.

For intervals $B = [b^L, b^U]$, $A = [a^L, a^U]$, the possibilities of $B \geq A$ and $A \geq B$ are defined in [9,10] as follows:

$$P(B \geq A) = \frac{\max\left\{0, b^U - a^L\right\} - \max\left\{0, b^L - a^U\right\}}{a^U - a^L + b^U - b^L}, \tag{16}$$

$$P(A \geq B) = \frac{\max\left\{0, a^U - b^L\right\} - \max\left\{0, a^L - b^U\right\}}{a^U - a^L + b^U - b^L}. \tag{17}$$

The similar expressions were proposed earlier by Facchinetti at al. [1] and by Xu and Da [11]. Xu and Chen [12] showed that the expressions proposed in [1,9] and [11] are equivalent ones.

A separate group of methods is based on the so-called probabilistic approach to interval comparison (see review in [7]). The idea to use the probability interpretation of interval is not a novel one. Nevertheless, only in [7] the complete consistent set of interval and fuzzy interval relations involving separated equality and inequality relations developed in the framework of probability approach is presented. Nevertheless, the results of interval comparison obtained using expressions (16) and (17) generally are similar to those obtained with the use of probabilistic approach to the interval comparison.

The main limitations of described above methods is that they provide an extent to which an interval is greater/lesser than another one if they have a common area (the intersection and inclusion cases should be considered separately [7]). If there are no intersections of compared intervals, the extent to which an interval is greater/lesser than another one is equal to 0 or 1 regardless of the distance between intervals. For example, Let A=[1,2], B=[3,4] and C=[100,200]. Then using described above approaches we obtain: $P(C > A)$=$P(B > A)$=1, $P(A > B)$=0.

Thus, we can say that in the case of overlapping intervals the above methods provide the possibility (or probability) that an interval is grater/lesser than another one and this possibility (or probability) can be treated as the strength of inequality or (in some sense) as the distance between compared intervals.

On the other hand, the above methods can not provide the measure of intervals inequality (distance) when they have no a common area.

Of course, the Hamming distance

$$d_H = \frac{1}{2}\left(\left|a^L - b^L\right| + \left|a^U - b^U\right|\right) \tag{18}$$

or Euclidean distance

$$d_E = \frac{1}{2}\left((a^L - b^L)^2 + (a^U - b^U)^2\right)^{\frac{1}{2}} \tag{19}$$

can be used as the distance between intervals, but these distances give no information about which interval is grater/lesser.

Hence, they can not be used directly for interval comparison especially when an interval is included into another one.

Therefore, here we propose to use directly the operation of interval subtraction [6] instead of Hamming and Euclidean distances. This method makes it possible to calculate the possibility (or probability) that an interval is grater/lesser that another one when they have a common area and when they do not intersect.

So for intervals $A = [a^L, a^U]$ and $B = [b^L, b^U]$, the result of subtraction is the interval $C{=}A - B{=}[c^L, c^U]$; $c^L = a^L - b^U$, $c^U = a^U - b^L$. It is easy to see that in the case of overlapping intervals A and B, we always obtain a negative left bound of interval C and a positive right bound. Therefore, to get a measure of distance between intervals which additionally indicate which interval is grater/lesser, we propose here to use the following value:

$$\Delta_{A-B} = \frac{1}{2}\left((a^L - b^U) + (a^U - b^L)\right). \tag{20}$$

It is easy to prove that for intervals with common center, Δ_{A-B} is always equal to 0. Really, expression (20) may be rewritten as follows:

$$\Delta_{A-B} = \left(\frac{1}{2}(a^L + a^U) - \frac{1}{2}(b^U + b^L)\right). \tag{21}$$

We can see that expression (21) represents the distance between the centers of compared intervals A and B. This is not a surprising result as Wang at al. [9] noted that most of the proposed methods for interval comparison are "totally based on the midpoints of interval numbers". It easy to see that the result of subtraction of intervals with common centers is an interval centered around 0. In the framework of interval analysis, such interval is treated as the interval 0.

More strictly, if a is a real value, then 0 can be defined as $a - a$. Similarly, if A is an interval, then interval zero may be defined as an interval $A - A{=}[a^L - a^U, a^U - a^L]$ which is centered around 0. Therefore, the value of Δ_{A-B} equal to 0 for A and B having a common center may be treated as a real valued representation of interval zero.

In Table 1, we present the values of $P(A \geq B)$, $P(B \geq A)$ (see expressions (16),(17)), the Hamming d_H and Euclidean d_H distances (see expressions

(18),(19)) between A_i and B and Δ_{A_i-B} for intervals $A_1 = [4, 7]$, $A_2 = [5, 8]$, $A_3 = [8, 11]$, $A_4 = [13, 16]$, $A_5 = [18, 21]$, $A_6 = [21, 24]$, $A_7 = [22, 25]$ and $B = [7, 22]$ placed as it is shown in Fig.1. The numbers in the first row in Table 1 correspond to the numbers of intervals A_i, i=1 to 7.

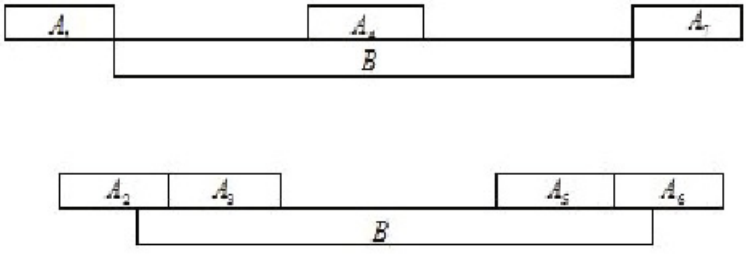


Fig. 1. Compared intervals

Table 1. Results of interval comparison

Method	1	2	3	4	5	6	7
$P(A \geq B)$	0	0.06	0.22	0.5	0.78	1	1
$P(A \leq B)$	1	0.94	0.78	0.5	0.22	0	0
d_E	10.82	10	7.81	6	7.81	10	10.82
d_H	9	8	6	6	6	8	9
Δ_{A-B}	-9	-8	-5	0	5	8	9

We can see that the values of Δ_{A_i-B} are negative when $A_i \leq B$ and become positive for $A_i \geq B$. These estimates coincide (at least qualitatively) with $P(A_i \geq B)$ and $P(B \geq A_i)$. So we can say that the sign of Δ_{A_i-B} indicates which interval is greater/lesser and the values of $abs(\Delta_{A_i-B})$ may be treated as the distances between intervals since these values are close the to the values of d_E and d_H in both cases: when intervals have a common area and when the there is no such an area.

3.3 The Comparison of the Direct Interval Extension of *TOPSIS* Method with the Know Method

Using Δ_{A-B}, it is easy to obtain from (14),(15) the ideal interval solutions

$$A^+ = \{[v_1^{+L}, v_1^{+U}], [v_2^{+L}, v_2^{+U}], ..., [v_1^{+L}, v_n^{+U}]\},$$

$$A^- = \{[v_1^{-L}, v_1^{-U}], [v_2^{-L}, v_2^{-U}], ..., [v_1^{-L}, v_n^{-U}]\}.$$

Since Δ_{A-B} is the subtraction of the midpoints of A and B, the values of S_i^+ and S_i^- may be calculated as follows:

$$S_i^+ = \frac{1}{2} \sum_{j \in K_B} \left((v_j^{+L} + v_j^{+U}) - (v_{ij}^L + v_{ij}^U) \right) + \\ + \frac{1}{2} \sum_{j \in K_C} \left((v_{ij}^L + v_{ij}^U) - (v_j^{+L} + v_j^{+U}) \right). \tag{22}$$

$$S_i^- = \frac{1}{2} \sum_{j \in K_B} \left((v_{ij}^L + v_{ij}^U) - (v_j^{-L} + v_j^{-U}) \right) + \\ + \frac{1}{2} \sum_{j \in K_C} \left((v_j^{-L} + v_j^{-U}) - (v_{ij}^L + v_{ij}^U) \right)). \tag{23}$$

Finally, using expression (13) we obtain the relative closeness RC_i to the ideal alternative.

Let us consider an illustrative example.

Suppose we deal with three alternatives A_i, i=1 to 3 and four local criteria C_j, j=1 to 4 presented by intervals in Table 2, where C_1 and C_2 are benefit criteria, C_3 and C_4 are cost criteria. Suppose, for the sake of simplicity that local criteria are of equal importance ($w_i = \frac{1}{n}$, i=1 to n). To stress the advantages of our method we chose the example in which many intervals representing the values of ratings intersect.

Table 2. Decision table

	C_1	C_2	C_3	C_4
A_1	[6, 22]	[10, 15]	[16, 21]	[18, 20]
A_2	[15, 18]	[8, 11]	[20, 30]	[19, 28]
A_3	[9, 13]	12, 17]	[42, 48]	[40, 49]

Then using the known method for interval extension of $TOPSIS$ method [3,4] (expressions (7)-(17)) we obtain R_1=0.5311, R_2=0.6378, R_3=0.3290 and therefore $R_2 > R_1 > R_3$, whereas with the use of our method (expressions (7), (8), (22), (23) and (13)) we get R_1=0.7688, R_2=0.7528, R_3=0.0717 and therefore $R_1 > R_2 > R_3$.

We can see that there is a considerable difference between the final ranking obtained by the known method and using our method based on the direct extension of $TOPSIS$ method. This can be explained by the fact that the method proposed in [3,4] has some limitations concerned with the presentation of intervals by real values in the calculation of ideal solutions and using the Euclidean distance when intervals intersect.

4 Conclusion

A new approach to the solution of $MCDM$ problems with the use of $TOPSIS$ method in the interval setting is proposed. This method called “direct interval

extension of $TOPSIS$ method" is free of the limitations of known method. It is shown that the presentation of intervals by real values in the calculation of ideal solutions used in the framework of known method may lead to the wrong results as well as the use of the Euclidean distance when intervals representing the values of local criteria intersect. Using the numerical example, it is shown that the proposed "direct interval extension of $TOPSIS$ method" may provide the final ranking of alternatives which is substantially differ from the results obtained using the known method.

References

1. Facchinetti, G., Ricci, R.G., Muzzioli, S.: Note on ranking fuzzy triangular numbers. International Journal of Intelligent Systems 13, 613–622 (1998)
2. Hwang, C.L., Yoon, K.: Multiple Attribute Decision Making Methods and Applications. Springer, Heidelberg (1981)
3. Jahanshahloo, G.R., Hosseinzade, L.F., Izadikhah, M.: An algorithmic method to extend TOPSIS for decision making problems with interval data. Applied Mathematics and Computation 175, 1375–1384 (2006)
4. Jahanshahloo, G.R., Hosseinzade, L.F., Izadikhah, M.: Extension of the TOPSIS method for decision making problems with fuzzy data. Applied Mathematics and Computation 181, 1544–1551 (2006)
5. Lai, Y.J., Liu, T.Y., Hwang, C.L.: TOPSIS for MODM. European Journal of Operational Research 76, 486–500 (1994)
6. Moore, R.E.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)
7. Sevastjanov, P.: Numerical methods for interval and fuzzy number comparison based on the probabilistic approach and DempsterShafer theory. Information Sciences 177, 4645–4661 (2007)
8. Wang, X., Kerre, E.E.: Reasonable properties for the ordering of fuzzy quantities (I) (II). Fuzzy Sets and Systems 112, 387–405 (2001)
9. Wang, Y.M., Yang, J.B., Xu, D.L.: A preference aggregation method through the estimation of utility intervals. Computers and Operations Research 32, 2027–2049 (2005)
10. Wang, Y.M., Yang, J.B., Xu, D.L.: A two-stage logarithmic goal programming method for generating weights from interval comparison matrices. Fuzzy Sets and Systems 152, 475–498 (2005)
11. Xu, Z., Da, Q.: The uncertain OWA operator. International Journal of Intelligent Systems 17, 569–575 (2002)
12. Xu, Z., Chen, J.: Some models for deriving the priority weights from interval fuzzy preference relations. European Journal of Operational Research 184, 266–280 (2008)
13. Yue, Z.: An extended TOPSIS for determining weights of decision makers with interval numbers. Knowledge-Based Systems 24, 146–153 (2011)

Enclosure for the Solution Set of Parametric Linear Systems with Non-affine Dependencies

Iwona Skalna

AGH University of Science and Technology, Krakow, Poland
skalna@agh.edu.pl

Abstract. The problem of solving linear systems whose coefficients are nonlinear functions of parameters varying within prescribed intervals is investigated. A new method for outer interval solution of such system is proposed. In order to reduce memory usage, nonlinear dependencies between parameters are handled using revised affine arithmetic. Some numerical experiments which aim to show the properties of the proposed method are reported.

Keywords: Parametric Linear Systems, Nonlinear Dependencies, Outer Interval Solution, Revised Affine Arithmetic.

1 Introduction

The problem of solving parametric linear systems arises in many scientific areas. When the parameters are uncertain but bounded and take arbitrary values from given intervals, a family of parametric linear systems is obtained. This family, called a *parametric interval linear system*, contains an infinite number of real linear systems. Therefore, solving the parametric interval linear system is to bound all possible solutions to the systems from the family.

Depending on how the parameters are involved in a linear system, parametric systems with *affine-linear dependencies* between the parameters and parametric linear systems with *nonlinear (non-affine) dependencies* between the parameters can be considered. Linear dependencies were investigated e.g. by Hladik [5], Popova [12], Rump [16], Shary [17] and Skalna [18]. Systems with nonlinear dependencies were considered e.g. by El-Owny [2], Popova [12] and Skalna [21]. Solving the latter involves computing an interval enclosure of the range of nonlinear functions over the domain of the parameters. To obtain tight range enclosures, generalised affine arithmetic was used in [2], generalised (extended) interval arithmetic was used in [12] and standard affine arithmetic was used in [21]. Generalised interval arithmetic provides sharp range enclosure for monotone functions. However, this methodology is not efficient for the general case. Affine arithmetic ([3]), revised affine arithmetic ([6], [7], [10]) or generalised affine arithmetic ([2], [4]) can be more appropriate in that case.

This paper presents a new method for computing an enclosure for the solution set of parametric linear systems involving arbitrary nonlinear dependencies.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 513–522, 2012.

Revised affine arithmetic, used to determine the bounds on the range of nonlinear functions, is introduced briefly in Section 4. Section 5 contains a general outline of the algorithm for computing an outer enclosure. Numerical examples presented in Section 6 illustrate the computational aspects of the proposed method. The paper ends with concluding remarks.

2 Parametric Interval Linear Systems

The following notations are used. Vectors are denoted by lower case italic letters and matrices are denoted by capital italic letters. Interval quantities are distinguished by writing them in bold. $\mathbb{R}^n$, $\mathbb{R}^{n\times m}$ denote, respectively, the set of real n-dimensional vectors and the set of real $n \times m$ matrices. A real compact interval $\boldsymbol{a} = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} \mid \underline{a} \leqslant x \leqslant \overline{a}\}$. The midpoint of an interval $\boldsymbol{a}$ is defined as $\check{a} = (\overline{a} + \underline{a})/2$ and the radius $\mathrm{r}(\boldsymbol{a}) = (\overline{a} - \underline{a})/2$. The set of all real compact intervals is denoted by $\mathbb{IR}$. Then, by $\mathbb{IR}^n$, $\mathbb{IR}^{n\times m}$ denote, respectively, the set of all n-dimensional interval vectors and the set of all interval $n \times m$ matrices. Square brackets are used to denote the range of a real function. Additionally, $\rho(\cdot)$ stands for a spectral radius.

Now consider a linear algebraic system

$$A(p)x(p) = b(p) \ ,$$

where $A(p) \in \mathbb{R}^{n\times n}$, $b(p) \in \mathbb{R}^n$, and $A_{ij}(p)$, $b_i(p)$ $(i, j = 1, \ldots, n)$ are nonlinear functions of a k-dimensional vector of parameters $p \in \mathbb{R}^k$.

When some of the parameters are assumed to be uncertain and vary within prescribed intervals $\boldsymbol{p}_i \ni p_i$ $(i = 1, \ldots, k)$, a family of parametric linear systems:

$$A(p)x(p) = b(p), \ p \in \boldsymbol{p} \ , \tag{1}$$

is obtained. This family is usually called a *parametric interval linear system*.

The set of solutions to all the systems from the family (1) is called a *parametric (united) solution set* and is defined as:

$$S_{\boldsymbol{p}} = S(A(p), b(p), \boldsymbol{p}) := \{x(p) \mid A(p)x(p) = b(p), \text{ for some } p \in \boldsymbol{p}\} . \tag{2}$$

In general, a parametric solution set has a very complicated structure, it does not even need to be convex. If $A(p)$ is non-singular for every $p \in \boldsymbol{p}$, then the parametric solution set is bounded. For a nonempty bounded set $S \in P\mathbb{R}^n$, the *interval hull*, that is the tightest interval vector containing S, is defined as

$$\Box S = \bigcap \{\boldsymbol{Y} \in \mathbb{IR}^n, \mid S \subseteq \boldsymbol{Y}\} \in \mathbb{IR}^n.$$

The interval hull of the parametric solution set is considered as an *(interval) hull solution* of the problem (1). It is quite expensive to obtain the hull solution, therefore an interval vector $\boldsymbol{x}^* \supseteq \Box S_{\boldsymbol{p}} \supseteq S_{\boldsymbol{p}}$, called an *outer interval solution*, is computed instead, and the goal for $\boldsymbol{x}^*$ is to be as narrow as possible.

3 Outer Interval Solution

It is easy to see that for arbitrary non-singular matrix $R \in \mathbb{R}^{n\times n}$ and vector $x_0 \in \mathbb{R}^n$

$$Ax = b \Leftrightarrow x - x_0 = (I - RA)(x - x_0) + R(b - Ax_0) \ . \tag{3}$$

Based on the above equivalence, a non-iterative method for outer interval enclosure of parametric interval linear system is proposed.

Theorem 1. *Consider parametric linear system (1). Let $R \in \mathbb{R}^{n\times n}$ be an arbitrary non-singular matrix and let $x_0 \in \mathbb{R}^n$. Denote $D = |I - [\{RA(p) \mid p \in \boldsymbol{p}\}]|$. If $\rho(D) < 1$ then*

$$S_{\boldsymbol{p}} \subseteq x_0 + (I - D)^{-1}|\,[\{R(b(p) - A(p)x_0) \mid p \in \boldsymbol{p}\}]\,|[-1,\,1].$$

Proof. Let $x \in S_{\boldsymbol{p}}$, then $A(p)x = b(p)$ for some $p \in \boldsymbol{p}$. From (3) it follows that

$$x - x_0 = (I - RA(p))(x - x_0) + R(b(p) - A(p)x_0),$$

which implies

$$\begin{aligned} |x - x_0| &= |(I - RA(p))(x - x_0) + R(b(p) - A(p)x_0)| \leqslant \\ &\leqslant |(I - RA(p))(x - x_0)| + |R(b(p) - A(p)x_0)| \leqslant \\ &\leqslant |(I - RA(p))||(x - x_0)| + |R(b(p) - A(p)x_0)|. \end{aligned} \tag{4}$$

The above inequality can be rewritten into

$$(I - |(I - RA(p))|)|x - x_0| \leqslant |R(b(p) - A(p)x_0)|. \tag{5}$$

Since $I - RA(p) \in I - [\{RA(p) \mid p \in \boldsymbol{p}\}]$, thus $|I - RA(p)| \leqslant D$ and, according to the theory of nonnegative matrices, $\rho(|I - RA(p)|) \leqslant \rho(D)$. If $\rho(D) < 1$, then $\rho(|I - RA(p)|) < 1$ and $(I - |I - RA(p)|)^{-1} \geqslant 0$. Thus, premultiplying both sides of (5) by $(I - |I - RA|)^{-1}$ yields

$$\begin{aligned} |x - x_0| &\leqslant (I - |I - RA(p)|)^{-1}|R(b(p) - A(p)x_0)| \leqslant \\ &\leqslant (I - D)^{-1}|\,[\{R(b(p) - A(p)x_0) \mid p \in \boldsymbol{p}\}]\,|, \end{aligned}$$

which means

$$x \in x_0 + (I - D)^{-1}|\,[\{R(b(p) - A(p)x_0) \mid p \in \boldsymbol{p}\}]\,|[-1,\,1].$$

□

Usually, the best choice for R is the numerically computed inverse of the midpoint matrix $\check{A}$ and $x_0 \approx R\check{b}$.

In what follows, the method for outer interval enclosure described by Theorem 1 will be called the M1 method. It can be proved that the M1 method coincides with the direct method presented in [21] if $R = \check{A}^{-1}$. Namely, one can show that if $R = \check{A}^{-1}$ then $I - |I - D| = \langle D \rangle$. Otherwise, those two methods produce different results.

Remark: The necessary condition for the M1 method, as well as for other similar methods, is that the spectral radius of the respective matrix is less than 1. Here, it is required that $\rho(|I - D|) < 1$. According to Proposition 4.1.1 from [11], the following equivalence holds: $\rho(|\check{A}^{-1}|\mathrm{r}(\boldsymbol{A})) < 1$ iff $\check{A}^{-1}\boldsymbol{A}$ is an H-matrix. Moreover, from Proposition 3.7.3 (see [11]) it follows that $\check{A}^{-1}\boldsymbol{A}$ is an H-matrix iff $\check{A}\langle\boldsymbol{A}\rangle$ is regular and $\check{A}\langle\boldsymbol{A}\rangle^{-1}e > 0$, where $e = (1, \ldots, 1)^T$. Since $\rho(|I - \check{A}\boldsymbol{A}|) < \rho(|\check{A}^{-1}|\mathrm{r}(\boldsymbol{A}))$, the latter condition can be useful to verify whether the assumption imposed on the spectral radius is fulfilled. Numerical experiments confirm the usefulness of this approach.

4 Revised Affine Arithmetic

The M1 method requires the enclosure of the range of the function $C(p) = RA(p)$ and $z(p) = R(b(p) - A(p)x_0)$ on the domain $\boldsymbol{p} \in \mathbb{IR}^k$. In this paper, the bounds on those ranges are computed using *revised affine arithmetic*.

Revised affine arithmetic (RAA) ([6], [7], [10]), [22]) is a modification of standard affine arithmetic (AA) ([3], [22]). RAA has lower memory requirements which is an important feature when solving large problems or during a long chain of computations. RAA keeps track of correlations between quantities, therefore it is able to provide much tighter bounds for the computed quantities than conventional interval arithmetic. In RAA, a partially unknown quantity x is represented by an affine form of a constant length

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x[-1,\, 1] \ ,$$

which consists of two parts: a first degree polynomial of length n and a cumulative error $e_x[-1,\, 1]$ ($e_x > 0$ is called an error variable) which represents the errors introduced by performing non-affine operations. In rigorous computations, it is also used to accumulate rounding errors inherent in floating-point arithmetic. The central value x_0, the coefficients x_i (called partial deviations) and e_x are finite floating-point numbers, and $\varepsilon_i \in [-1,\, 1]$ are dummy variables. The radius of a revised affine form $\mathrm{r}(\hat{x}) = \sum_{i=1}^{n} |x_i| + e_x$.

All the standard arithmetic operations as well as other classical functions are redefined for revised affine forms, so that they result straightforwardly in affine forms. At the end of any computation, the resulting affine form $\hat{x}$ is transformed into the interval $[\hat{x}] = [x_0 - \mathrm{r}(\hat{x}),\, x_0 + \mathrm{r}(\hat{x})]$ which is the tightest interval that contains all possible values of $\hat{x}$, assuming that each ε_i varies independently over its domain [22].

5 Algorithm and Implementation Issues

The M1 method is described in Algorithm 1. The overall computation cost is $O(n^3)$. The computations were performed using the author's own software which implements operations on intervals and on affine forms using upward rounding only. This allows one to reduce computation time.

Algorithm 1. (Method for outer interval solution)

1. $\check{A} = A(\check{p})$; $\check{b} = b(\check{p})$
2. $R \approx \check{A}^{-1}$; $x_0 = R\check{b}$
3. $D = |I - \square\{RA(p) \mid p \in \boldsymbol{p}\}|$
4. $z = |\square\{R(b(p) - A(p)\boldsymbol{x}_0)\}|$
5. $G \approx (I - D)^{-1}$
6. $\boldsymbol{x}^* = \boldsymbol{x}_0 + G \cdot z[-1, 1]$

In order to further reduce the width of the result obtained using the above algorithm, parametric version of the well known Jacobi iteration is applied.

Remark: As can be seen from Algorithm 1, the method requires calculating the inverses of $\check{A}$ and M. In fact, some numerically computed approximations $R \approx \check{A}$ and $G \approx M$ are obtained. To find out how the approximation errors influence the final result, a more time-consuming version of the algorithm can be used. It differs from the method presented in Algorithm 1 in that the real matrices $\check{A}$ and M are converted into interval ones $A \to \boldsymbol{A}$, $M \to \boldsymbol{M}$, and then their interval inverses are used in computations.

6 Numerical Experiments

In this section, the quality of the solution enclosures given by the new M1 method is presented. The results of the M1 method are compared to the results obtained using the following methods: Rump's parametric fixed-point iteration (RPFPI) ([2], [12]), Bauer-Skeel (BS) method [5], Hansen-Bliek-Rohn method [5] and Interval-Affine Gaussian Elimination [1]. All of them are polynomial complexity methods for solving parametric interval linear systems. The hull solution (computed using global optimisation methods [19], [20]) is given as a reference solution. The percentage by which the hull solution is overestimated by outer enclosure is measured as $\mathcal{O}_M = 100 \cdot (1 - \mathrm{r}(S_{\boldsymbol{p}})/\mathrm{r}(\boldsymbol{x}^*_M))$, where M stand for the name of the respective method. For clarity, the results presented below include the result given by the M1 method and the best result obtained using the remaining methods.

Example 1 (Three-dimensional parametric linear system)

$$\begin{pmatrix} -(p_1+1)p_2 & p_1^2(p_3-p_4) & -p_2 \\ p_5/\sqrt{p_2 p_4} & p_2(p_2-p_3) & 1 \\ p_1 p_2 & (p_1-p_3)p_5 & \sqrt{p_2} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} p_1 \\ p_1^2 - p_2 p_3 \\ -2p_3 \end{pmatrix} . \tag{6}$$

All the parameters are considered to be uncertain with uncertainty ranging from 2% to 18% in their nominal values: $p_1 = 1.2$, $p_2 = 2.2$, $p_3 = 0.51$, $p_4 = p_5 = 0.4$. The results, given to 5 decimals, obtained for selected uncertainties are presented in the Table 1. The summary of the results obtained for the entire range $(2\% - 18\%)$ of uncertainty is depicted in Fig. 1.

Table 1. Solution of the system (6) with 2%, 6%, 10% and 18% uncertainty in all parameters

x	$\boldsymbol{x}^*_{M1}$	$\boldsymbol{x}^*_{BS}$	$\square S_{\boldsymbol{p}}$	$\mathcal{O}_{M1}$	$\mathcal{O}_{BS}$
2% uncertainty					
x_1	[0.61255, 0.73299]	[0.60965, 0.73352]	[0.62032, 0.72918]	9.6%	12.1%
x_2	[0.49967, 0.58609]	[0.49751, 0.58663]	[0.50396, 0.58416]	7.2%	10.0%
x_3	[−2.12049, −1.853]	[−2.12099, −1.84683]	[−2.11430, −1.86828]	8.0%	10.3%
6% uncertainty					
x_1	[0.32168, 1.06148]	[0.26428, 1.07946]	[0.49552, 0.95536]	37.8%	43.6%
x_2	[0.31014, 0.80559]	[0.26980, 0.81361]	[0.40842, 0.74086]	32.9%	38.9%
x_3	[−2.81291, −1.24995]	[−2.83450, −1.13459]	[−2.62992, −1.58915]	33.4%	38.8%
10% uncertainty					
x_1	[0.14222, 1.26479]	[0.03637, 1.30772]	[0.46134, 1.05614]	47.0%	53.2%
x_2	[0.20138, 0.92475]	[0.12818, 0.95480]	[0.38165, 0.80640]	41.3%	48.6%
x_3	[−3.22753, −0.89186]	[−3.28554, −0.68430]	[−2.86095, −1.51343]	42.3%	48.2%
18% uncertainty					
x_1	[−6.45933, 8.04476]	[−11.06525, 12.41151]	[0.34568, 1.71831]	90.5%	94.2%
x_2	[−3.62232, 4.87106]	[−6.36777, 7.44823]	[0.29313, 1.17828]	89.6%	93.6%
x_3	[−16.51497, 11.97641]	[−24.46694, 20.49209]	[−4.38845, −1.27018]	89.1%	93.1%

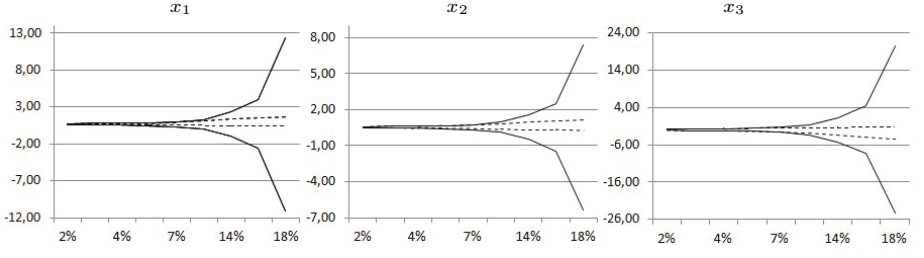


Fig. 1. Summary of the results for system (6) (uncertainty: 2% − 18%)

As can be seen, the overestimation of the enclosure increases rapidly with increase of the width of parameter intervals. This is probably due to the poor enclosure of the range of the respective nonlinear functions. This in turn follows from the fact that in an RAA computation, the error term produced by nonlinear operations became more important as the range of the operands became wider. Nevertheless, it is worth noting that the M1 method always yield better results than the remaining methods considered here.

Example 2 (Simple planar frame). Consider a simple planar frame with three support types and external load uniformly distributed along the horizontal beam (see e.g. [9], [14]). Similar to the previous example, all the parameters are considered to be uncertain. The problem is solved with uncertainty ranging from 1% to 70%. The parametric linear system $Ax = b$ describing the frame is given by the following relations (elements not listed below are equal to zero):

$$\begin{aligned}
&\tfrac{1}{2}a_{11} = a_{12} = a_{21} = a_{65} = -a_{74} = l_{12} \\
&a_{22} = 2l_{12} + 2l_{23},\ a_{33} = 3l_{24} + 2l_{23},\ a_{66} = l_{12} + l_{24} \\
&a_{23} = a_{32} = -2l_{23},\ a_{68} = l_{23},\ a_{86} = l_{24} \\
&a_{47} = a_{48} = a_{54} = a_{55} = a_{56} = -a_{61} = -a_{71} = a_{72} = -a_{83} = 1 \\
&b = \left(0,\ 0,\ -\tfrac{3}{8}ql_{24}^3,\ 0,\ ql_{24},\ ql_{24}(l_{12} + \tfrac{1}{2}l_{24},\ 0,\ \tfrac{1}{2}ql_{24}^2\right) \ .
\end{aligned} \tag{7}$$

Table 2 lists the results for moments and reactions of the planar frame system with: a) 1% uncertainty in all parameters, b) 2% uncertain lengths and 30% uncertain load and c) 20% uncertainty in all parameters. In case a), the results show a good sharpness of the enclosures for most of the solution components. Only the two last components (horizontal reactions R_1^x and R_3^x) are more

Table 2. Solutions for moments and reactions for the planar frame system with: a) 1% uncertainty in all parameters, b) 2% uncertain lengths and 30% uncertain load and c) 20% uncertainty in all parameters

x	$\boldsymbol{x}^*_{M1}$	$\boldsymbol{x}^*_{BS}$	$\square S\boldsymbol{p}$	$\mathcal{O}_{M1}$	$\mathcal{O}_{BS}$
a) 1% uncertainty in all parameters					
M_1	[0.24466, 0.25533]	[0.24466, 0.25537]	[0.24479, 0.25529]	1.6%	2.0%
M_{21}	[−0.51063, −0.48936]	[−0.51070, −0.48936]	[−0.51059, −0.48959]	1.3%	1.6%
M_{24}	[−1.01720, −0.98285]	[−1.01726, −0.98286]	[−1.01710, −0.98310]	1.0%	1.2%
R_1^y	[−0.76980, −0.73026]	[−0.76990, −0.73019]	[−0.76973, −0.73072]	1.3%	1.8%
R_3^y	[6.66853, 6.83165]	[6.66828, 6.83180]	[6.66989, 6.83089]	1.3%	1.5%
R_4^y	[3.95927, 4.04062]	[3.95929, 4.04072]	[3.96010, 4.04010]	1.7%	1.8%
R_1^x	[−0.68732, −0.64652]	[−0.68751, −0.64636]	[−0.68420, −0.64953]	15.0%	15.7%
R_3^x	[0.64651, 0.68732]	[0.64636, 0.68751]	[0.64953, 0.68420]	15.0%	15.7%
b) 2% uncertain lengths and 30% uncertain load					
M_1	[0.20186, 0.29807]	[0.20185, 0.29830]	[0.20578, 0.29681]	5.4%	5.6%
M_{21}	[−0.59541, −0.40449]	[−0.59583, −0.40447]	[−0.59361, −0.41155]	4.6%	4.9%
M_{24}	[−1.18059, −0.81957]	[−1.18103, −0.81957]	[−1.17778, −0.82973]	3.6%	3.7%
R_1^y	[−0.90217, −0.59795]	[−0.90277, −0.59768]	[−0.89941, −0.61122]	5.3%	5.5%
R_3^y	[5.59427, 7.90650]	[5.59316, 7.90739]	[5.65854, 7.87074]	4.3%	4.4%
R_4^y	[3.33125, 4.66787]	[3.33120, 4.66870]	[3.36600, 4.64600]	4.2%	4.3%
R_1^x	[−0.93187, −0.40482]	[−0.93292, −0.40414]	[−0.79948, −0.54331]	51.4%	51.6%
R_3^x	[0.40476, 0.93187]	[0.40414, 0.93292]	[0.54331, 0.79948]	51.4%	51.6%
c) 20% uncertainty in all parameters					
M_1	[0.09780, 0.39890]	[0.09516, 0.41640]	[0.16081, 0.37349]	29.4%	33.8%
M_{21}	[−0.77776, −0.21966]	[−0.80657, −0.21655]	[−0.74699, −0.32162]	23.8%	27.9%
M_{24}	[−1.43620, −0.58421]	[−1.45800, −0.58825]	[−1.38123, −0.69798]	19.8%	21.4%
R_1^y	[−1.29885, −0.22341]	[−1.34574, −0.18894]	[−1.24498, −0.43857]	25.0%	30.3%
R_3^y	[4.67728, 8.89420]	[4.56259, 8.96834]	[5.28269, 8.52968]	23.0%	26.3%
R_4^y	[2.85445, 5.09986]	[2.86117, 5.14258]	[3.24000, 4.84000]	28.7%	29.9%
R_1^x	[−2.66291, 1.12482]	[−2.77403, 1.22653]	[−1.10665, −0.38984]	81.1%	82.1%
R_3^x	[−1.13239, 2.66291]	[−1.22653, 2.77403]	[0.38984, 1.10665]	81.1%	82.1%

sensitive to the variations of the parameters. Solutions for moments and reactions of the planar frame system with 2% uncertain lengths and 30% uncertain load (case b)) look quite reasonable despite the large uncertainty in the load. Except for the very sensitive components of the horizontal reactions R_1^x and R_3^x, the overestimation for other solution components has increased by only about three and a half times. In this case, the result of the M1 method is very similar to the best result produced by the remaining methods. Finally, in case c), it can be seen that the overestimation is still acceptable; the uncertainty of the result, except for the horizontal reactions R_1^x and R_3^x which, as already shown, are very sensitive to the variations of the parameters, is at the similar level as the uncertainty of the parameters. In this case, the result of the M1 method is noticeably better than the results of the remaining methods.

The summary of the results obtained for the entire range (1% – 70%) of uncertainty is shown in Figure 2. One can see that the overestimation starts to grow rapidly for uncertainties larger than 40%.

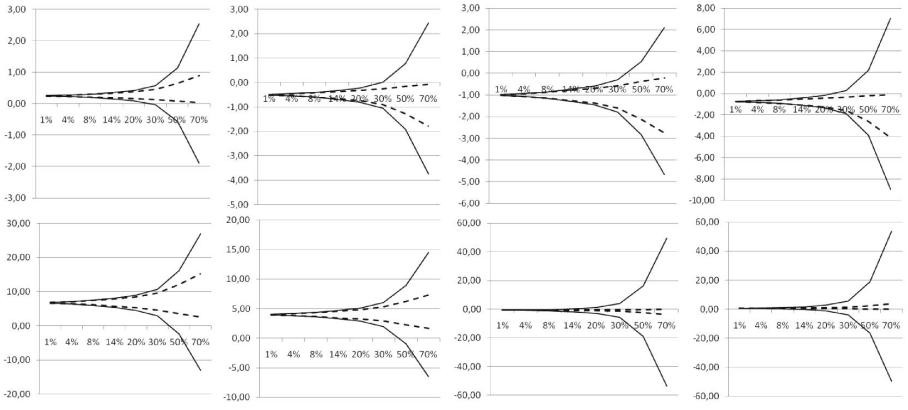


Fig. 2. Summary of the results for planar frame (uncertainty: 1% – 70%)

7 Conclusions

A new method M1 for solving parametric interval linear systems with nonlinear dependencies was suggested in Section 3. Revised affine arithmetic was used to compute ranges of nonlinear functions, required by the M1 method. The M1 method was compared with other, polynomial complexity, methods for solving parametric interval linear systems, including Rump's parametric fixed point iteration, to evaluate its performance. It turned out to be the best among all considered methods.

Moreover, it was demonstrated that the M1 produce sharp enclosures for small parameter uncertainties. However, the overestimation increases, sometimes very rapidly, with increasing the width of parameter intervals. The reason for this is

that in affine arithmetic an approximation error of nonlinear operations increases as the ranges of the operands get wider. To obtain sharper enclosures one may subdivide parameter intervals or use other tools for bounding ranges. Which way to choose requires additional study for efficiency and accuracy.

References

1. Akhmerov, R.R.: Interval-Affine Gaussian Algorithm for Constrained Systems. Reliable Computing 11(5), 323–341 (2005)
2. 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, vol. 16 (2006)
3. de Figueiredo, L.H., Stolfi, J.: An Introduction to Affine Arithmetic. TEMA Tend. Mat. Apl. Comput. 4(3), 297–312 (2003)
4. Hansen, E.R.: Generalized Interval Arithmetic. In: Nickel, K. (ed.) Interval Mathematics. LNCS, vol. 29, pp. 7–18. Springer, Heidelberg (1975)
5. Hladik, M.: Enclosures for the solution set of parametric interval linear systems. Technical Report 983, KAM-DIMATIA Series, pp. 1–25 (2010)
6. Vu, X.-H., Sam-Haroud, D., Faltings, B.: A Generic Scheme for Combining Multiple Inclusion Representations in Numerical Constraint Propagation. Technical Report No. IC/2004/39, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland (2004)
7. Kolev, L.V.: Automatic Computation of a Linear Interval Enclosure. Reliable Computing 7(1), 17–28 (2001)
8. Kolev, L.V.: Solving Linear Systems whose Elements are Non-linear Functions of Intervals. Numerical Algorithms 37, 213–224 (2004)
9. 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), http://andrzej.pownuk.com/publications/IntervalEquations.pdf
10. Messine, F.: Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization. Journal of Universal Computer Science 8(11), 992–1015 (2002)
11. Neumaier, A.: Interval Methods for Systems of Equations, pp. xvi–255. Cambridge University Press, Cambridge (1990)
12. Popova, E.D.: Generalization of a Parametric Fixed-Point Iteration. PAMM 4(1), 680–681 (2004)
13. Popova, E.D.: On the Solution of Parametrised Linear Systems. In: Kraemer, W., Wolff von Gudenberg, J. (eds.) Scientific Computing, Validated Numerics, Interval Methods, pp. 127–138. Kluwer Acad. Publishers (2001)
14. 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)
15. Rohn, J.: Technical Report No. 620, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, p. 11 (1995)
16. Rump, S.M.: Verification methods for dense and sparse systems of equations. In: Herzberger, J. (ed.) Topics in Validated Computations, pp. 63–135. North-Holland, Amsterdam (1994)
17. Shary, S.P.: Solving tied interval linear systems. Sibirskii Zhurnal Vychislitiel'noi Matiematiki 7(4), 363–376 (2004)

18. 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)
19. 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)
20. Skalna, I., Pownuk, A.: A global optimisation method for computing interval hull solution for parametric linear systems. International Journal of Reliability and Safety 3(1-3), 235–245 (2009)
21. Skalna, I.: Direct Method for Solving Parametric Interval Linear Systems with Non-affine Dependencies. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009. LNCS, vol. 6068, pp. 485–494. Springer, Heidelberg (2010)
22. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: Proc. SIBGRAPI 1993, VI Simpsio Brasileiro de Computaa o Grfica e Processamento de Imagens, Recife, BR, pp. 9–18 (1993)

The Central Difference Interval Method for Solving the Wave Equation

Barbara Szyszka

Poznan University of Technology
Institute of Mathematics
Piotrowo 3A, 60-965 Poznan, Poland
Barbara.Szyszka@put.poznan.pl

Abstract. A way of constructing the interval method of second order for solving one dimensional wave equation is presented in the paper. The central difference interval method for the hyperbolic Partial Differential Equation is taken into consideration. The suitable Dirichlet and Cauchy conditions are satisfied for the string with fixed endpoints. The estimations of discretization errors are proposed. The method of floating-point interval arithmetic is studied. The numerical experiment is presented.

Keywords: Interval method, Floating-Point Interval Arithmetic, Wave equation, Difference method, Initial-boundary value problems.

1 Introduction

Interval methods for solving the initial value problem in Ordinary Differential Equations (ODE) have been presented for example in papers [1], [2], [3] and [4]. The studies on ODE have been extended to Partial Differential Equations (PDE) in context to interval methods of floating point interval arithmetic.

The paper is devoted to the central difference interval method for solving PDE together with boundary and initial conditions. The main point of construction interval methods is to contain all numerical errors into obtained solutions. The backward and central difference interval methods for solving hyperbolic PDE were presented at GAMM [5] and ICNAAM [6] Conferences. Both methods concern the error of an initial condition of order $O(h)$ and the estimations of errors which are depended on a velocity v (a parameter of the wave equation). The another way of errors estimation of discretization method is presented in the paper. It is assumed that an initial condition with local truncation error is $O(h^4)$ and estimations of errors do not depend on velocity v.

2 The Wave Equation

The string, as an example of the one dimensional wave equation, is taken into consideration [7] and [8], with following assumptions:

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 523–532, 2012.

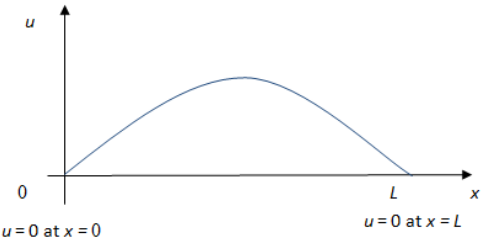


Fig. 1. The vibrating string

- the string is well flexible and homogeneous (mass of string per unit length ϱ is a constant),
- the tension T of the string is constant and larger than the force of gravity (no other external forces act on the string),
- damping effects are neglected,
- the amplitude is not too large,
- each inside point of the string can move only in the vertical direction.

If the string is stretched between two points (see Fig. 1) where $x = 0$ and $x = L$ and $u = u(x,t)$ denotes the amplitude of the string's displacement, then u satisfies the wave equation

$$v^2 \frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\partial^2 u(x,t)}{\partial t^2} = 0, \tag{1}$$

in the region where $0 < x < L$ and time $t > 0$, where $v^2 = \frac{T}{\varrho} = const.$

Since the ends of string are secured to the x-axis, u must also satisfy the Dirichlet boundary conditions, for $t > 0$, given by

$$\begin{aligned} u(0,t) &= 0, \\ u(L,t) &= 0, \end{aligned} \tag{2}$$

The initial position and velocity of the string are given by the Cauchy initial conditions at the time $t = 0$

$$\begin{aligned} u(x,0) &= \varphi(x) \\ \left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0} &= \psi(x). \end{aligned} \tag{3}$$

for $0 < x < L$, where $\varphi(x)$ and $\psi(x)$ are given functions.

3 The Central Difference Method

To set up the central difference method the positive numbers n and m, which adequately describe number of points x and t, are selected. The space-time grid

with the mesh points (x_i, t_j) is received, where $\Delta x = h > 0$ and $\Delta t = k > 0$ such that

$$\begin{aligned} x_i &= i \cdot h, \quad h = \frac{L}{n} > 0, \quad i = 0, 1, \ldots, n, \\ t_j &= j \cdot k, \quad k > 0, \quad j = 0, 1, \ldots, m. \end{aligned} \tag{4}$$

The Taylor series for each interior mesh point (x_i, t_j) are used to obtain the central difference formulas

$$\begin{aligned} \frac{\partial^2 u(x_i, t_j)}{\partial x^2} &= \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j)}{h^2} - \frac{h^2}{12} \cdot \frac{\partial^4 u(\xi_i, t_j)}{\partial x^4}, \\ \frac{\partial^2 u(x_i, t_j)}{\partial t^2} &= \frac{u(x_i, t_{j+1}) - 2u(x_i, t_j) + u(x_i, t_{j-1})}{k^2} - \frac{k^2}{12} \cdot \frac{\partial^4 u(x_i, \eta_j)}{\partial t^4}, \end{aligned} \tag{5}$$

where $\xi_i \in (x_{i-1}, x_{i+1})$ and $\eta_j \in (t_{j-1}, t_{j+1})$, for $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, m-1$.

Substituting the formulas (5) into the wave equation (1), gives

$$\begin{aligned} 2\left(v^2 \frac{k^2}{h^2} - 1\right) u(x_i, t_j) + u(x_i, t_{j-1}) + u(x_i, t_{j+1}) - \\ -v^2 \frac{k^2}{h^2} \left(u(x_{i-1}, t_j) + u(x_{i+1}, t_j)\right) = \frac{k^4}{12} \cdot \frac{\partial^4 u(x_i, \eta_j)}{\partial t^4} - v^2 \frac{h^2 k^2}{12} \cdot \frac{\partial^4 u(\xi_i, t_j)}{\partial x^4}. \end{aligned} \tag{6}$$

Neglecting the truncation errors in the formula (6) the central difference method is generated

$$\begin{aligned} 2\left(v^2 \frac{k^2}{h^2} - 1\right) u(x_i, t_j) + u(x_i, t_{j-1}) + u(x_i, t_{j+1}) - \\ -v^2 \frac{k^2}{h^2} \left(u(x_{i-1}, t_j) + u(x_{i+1}, t_j)\right) = 0, \end{aligned} \tag{7}$$

for $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, m-1$.

The boundary conditions (2), for $j = 1, 2, \ldots, m-1$, give

$$\begin{aligned} u(x_0, t_j) &= 0, \\ u(x_n, t_j) &= 0. \end{aligned} \tag{8}$$

The initial conditions (3), for $i = 1, 2, \ldots, n-1$, we can write as follows

$$\begin{aligned} u(x_i, t_0) &= \varphi(x_i), \\ u(x_i, t_1) &= \varphi(x_i) + k\psi(x_i) + \frac{k^2}{2} v^2 \varphi''(x_i) + \frac{k^3}{6} v^3 \varphi'''(x_i), \end{aligned} \tag{9}$$

To obtain the second equation in the formula (9) the derivative

$$\left. \frac{\partial u(x_i, t_j)}{\partial t} \right|_{t=0}$$

by the forward difference approximation of third order the Taylor series in the variable t about t_1 for points (x_i, t_1) is replaced

$$u(x_i,t_1) = u(x_i,t_0) + k \cdot \frac{\partial u(x_i,t_0)}{\partial t} + \frac{k^2}{2} \cdot \frac{\partial^2 u(x_i,t_0)}{\partial t^2} + \\ + \frac{k^3}{6} \cdot \frac{\partial^3 u(x_i,t_0)}{\partial t^3} + \frac{k^4}{24} \cdot \frac{\partial^4 u(x_i,\tilde{\eta}_i)}{\partial t^4}, \tag{10}$$

where $\tilde{\eta}_i \in (t_0, t_1)$ for $i = 1, 2, \ldots, n-1$.

If the first equation in the formula (9) is differentiated twice in respect to x (see [9]) then we can write

$$\frac{\partial^2 u(x_i,t_0)}{\partial t^2} = v^2 \frac{\partial^2 u(x_i,t_0)}{\partial x^2} = v^2 \frac{d^2\varphi(x_i)}{dx^2} = v^2 \varphi(x_i)'',$$

and we have

$$\frac{\partial^2 u(x_i,t_0)}{\partial t^2} = v^2 \varphi(x_i)'', \tag{11}$$

Analogously, if $\frac{d^3\varphi(x_i)}{dx^3}$ exists, then we can write

$$\frac{\partial^3 u(x_i,t_0)}{\partial t^3} = v^3 \varphi(x_i)'''. \tag{12}$$

Substituting (11), (12) and the first equation (9) into formula (10), gives

$$u(x_i,t_1) = \varphi(x_i) + k\psi(x_i) + \frac{k^2}{2} v^2 \varphi''(x_i) + \frac{k^3}{6} v^3 \varphi'''(x_i) + \frac{k^4}{24} \cdot \frac{\partial^4 u(x_i,\tilde{\eta}_i)}{\partial t^4} \tag{13}$$

where $\tilde{\eta}_i \in (t_0, t_1)$ for $i = 1, 2, \ldots, n-1$.

Neglecting the truncation error, the second dependence of the formula (9) is obtained.

4 The Central Difference Interval Method

In conventional difference methods the truncation errors frequently are neglected. The idea of the interval methods is to involve these errors into interval solution.

We consider the central difference method (7) and the second initial condition (9) together with their truncations errors.

The truncation error E_M of method is in the form

$$E_M = \frac{k^4}{12} \cdot \frac{\partial^4 u(x_i,\eta_j)}{\partial t^4} - v^2 \frac{k^2 h^2}{12} \cdot \frac{\partial^4 u(\xi_i,t_j)}{\partial x^4}, \tag{14}$$

where
$\xi_i \in (x_{i-1}, x_{i+1})$ for $i = 1, 2, \ldots, n-1$,
$\eta_j \in (t_{j-1}, t_{j+1})$ for $j = 1, 2, \ldots, m-1$.

The approximation error E_C of the second initial condition (9) is defined as

$$E_C = \frac{k^4}{24} \cdot \frac{\partial^4 u(x_i,\tilde{\eta}_i)}{\partial t^4}, \tag{15}$$

where
$\tilde{\eta}_i \in (t_0, t_1)$ for $i = 1, 2, \ldots, n-1$.

To obtain the interval method the estimations of truncations errors E_M and E_C are needed. To obtain estimations of $\frac{\partial^4 u(\xi_i, t_j)}{\partial x^4}$ and $\frac{\partial^4 u(x_i, \eta_j)}{\partial t^4}$ in (14), the wave equation (1) twice with respect to x and twice with respect to t is differentiated:

$$\frac{\partial^4 u(x_i, t_j)}{\partial x^4} = \frac{1}{v^2} \cdot \frac{\partial^4 u(x_i, t_j)}{\partial x^2 \partial t^2}, \quad \frac{\partial^4 u(x_i, t_j)}{\partial t^4} = v^2 \frac{\partial^4 u(x_i, t_j)}{\partial x^2 \partial t^2}. \tag{16}$$

for $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, m-1$.

Let

$$\left| \frac{\partial^4 u(x_i, t_j)}{\partial x^2 \partial t^2} \right| \leq M$$

for each mesh point (x_i, t_j), where $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, m$.

Then

$$\frac{\partial^4 u(x_i, t_j)}{\partial x^2 \partial t^2} \in [-M, M]. \tag{17}$$

The relation (17) into formulas (16) is put as follows

$$\frac{\partial^4 u(x_i, t_j)}{\partial x^4} \in \frac{1}{v^2}[-M, M], \quad \frac{\partial^4 u(x_i, t_j)}{\partial t^4} \in v^2[-M, M],$$

Thus, these relations for $\xi_i \in (x_{i-1}, x_{i+1})$, $\eta_j \in (t_{j-1}, t_{j+1})$, where $i = 0, 1, \ldots, n$ and $j = 1, 2, \ldots, m-1$ are in the forms

$$\begin{aligned} \frac{\partial^4 u(\xi_i, t_j)}{\partial x^4} &\in \frac{1}{v^2}[-M, M], \\ \frac{\partial^4 u(x_i, \eta_j)}{\partial t^4} &\in v^2[-M, M]. \end{aligned} \tag{18}$$

Putting the relations (18) into (14) gives

$$E_M \in \frac{k^2}{12}(h^2 + k^2 v^2)[-M, M]. \tag{19}$$

The derivative $\frac{\partial^4 u(x_i, \tilde{\eta}_i)}{\partial t^4}$ in (15) for $\tilde{\eta}_i \in (t_0, t_1), i = 1, 2, \ldots, n-1$, is a special case of derivative $\frac{\partial^4 u(x_i, \eta_j)}{\partial t^4}$ in (14) for $\eta_j \in (t_{j-1}, t_{j+1}), j = 1, 2, \ldots, m-1$. Then the estimation of $\frac{\partial^4 u(x_i, \eta_j)}{\partial t^4}$ is true for $\frac{\partial^4 u(x_i, \tilde{\eta}_i)}{\partial t^4}$. Putting the second relation (18) into (15) gives

$$E_C \in \frac{k^4 v^2}{24}[-M, M]. \tag{20}$$

To obtain the estimations of truncations errors E_M (19) and E_C (20) the value M is needed. For the purpose of estimation value M the Taylor series for the formulas (5) are used as follows

$$\frac{\partial^2}{\partial t^2}\left(\frac{\partial^2 u(x_i,t_j)}{\partial x^2}\right)=\frac{u(x_{i+1},t_{j+1})-2u(x_{i+1},t_j)+u(x_{i+1},t_{j-1})}{h^2k^2}-$$
$$-\frac{k^2}{12h^2}\cdot\frac{\partial^4 u(x_{i+1},\hat{\eta}_j)}{\partial t^4}-$$
$$-2\Big(\frac{u(x_i,t_{j+1})-2u(x_i,t_j)+u(x_i,t_{j-1})}{h^2k^2}-\frac{k^2}{12h^2}\cdot\frac{\partial^4 u(x_i,\tilde{\eta}_j)}{\partial t^4}\Big)+ \tag{21}$$
$$+\frac{u(x_{i-1},t_{j+1})-2u(x_{i-1},t_j)+u(x_{i-1},t_{j-1})}{h^2k^2}-\frac{k^2}{12h^2}\cdot\frac{\partial^4 u(x_{i-1},\check{\eta}_j)}{\partial t^4}-$$
$$-\frac{h^2}{12}\cdot\frac{\partial^2}{\partial t^2}\left(\frac{\partial^4 u(\xi_i,t_j)}{\partial x^4}\right),$$

and

$$\frac{\partial^2}{\partial x^2}\left(\frac{\partial^2 u(x_i,t_j)}{\partial t^2}\right)=\frac{u(x_{i+1},t_{j+1})-2u(x_i,t_{j+1})+u(x_{i-1},t_{j+1})}{k^2h^2}-$$
$$-\frac{h^2}{12k^2}\cdot\frac{\partial^4 u(\hat{\xi}_i,t_{j+1})}{\partial x^4}-$$
$$-2\Big(\frac{u(x_{i+1},t_j)-2u(x_i,t_j)+u(x_{i-1},t_j)}{k^2h^2}-\frac{h^2}{12k^2}\cdot\frac{\partial^4 u(\tilde{\xi}_i,t_j)}{\partial x^4}\Big)+ \tag{22}$$
$$+\frac{u(x_{i+1},t_{j-1})-2u(x_i,t_{j-1})+u(x_{i-1},t_{j-1})}{k^2h^2}-\frac{h^2}{12k^2}\cdot\frac{\partial^4 u(\check{\xi}_i,t_{j-1})}{\partial x^4}-$$
$$-\frac{k^2}{12}\cdot\frac{\partial^2}{\partial x^2}\left(\frac{\partial^4 u(x_i,\eta_j)}{\partial t^4}\right),$$

where $\eta_j\,\hat{\eta}_j,\tilde{\eta}_j,\check{\eta}_j\in(t_{j-1},t_{j+1})$ for $j=1,2,\ldots,m-1$,
and $\xi_i,\hat{\xi}_i,\tilde{\xi}_i,\check{\xi}_i\in(x_{i-1},x_{i+1})$ for $i=1,2,\ldots,n-1$.

Let

$$\frac{\partial^4 u(x_i,t_j)}{\partial x^2\partial t^2}=\frac{\partial^4 u(x_i,t_j)}{\partial t^2\partial x^2}$$

together with (17) for each mesh point (x_i,t_j), where $i=0,1,\ldots,n$ and $j=0,1,\ldots,m$. Neglecting truncations errors in formulas (21) and (22) we can write them as

$$\frac{\partial^2}{\partial t^2}\Big(\frac{\partial^2 u(x_i,t_j)}{\partial x^2}\Big)\simeq\frac{u(x_{i+1},t_{j+1})-2u(x_{i+1},t_j)+u(x_{i+1},t_{j-1})}{h^2k^2}-$$
$$-2\frac{u(x_i,t_{j+1})-2u(x_i,t_j)+u(x_i,t_{j-1})}{h^2k^2}+ \tag{23}$$
$$+\frac{u(x_{i-1},t_{j+1})-2u(x_{i-1},t_j)+u(x_{i-1},t_{j-1})}{h^2k^2}$$

and

$$\frac{\partial^2}{\partial x^2}\Big(\frac{\partial^2 u(x_i,t_j)}{\partial t^2}\Big)\simeq\frac{u(x_{i+1},t_{j+1})-2u(x_i,t_{j+1})+u(x_{i-1},t_{j+1})}{k^2h^2}-$$
$$-2\frac{u(x_{i+1},t_j)-2u(x_i,t_j)+u(x_{i-1},t_j)}{k^2h^2}+ \tag{24}$$
$$+\frac{u(x_{i+1},t_{j-1})-2u(x_i,t_{j-1})+u(x_{i-1},t_{j-1})}{k^2h^2}$$

The right hand sides of the approximations (23) and (24) are the same, then constance M we can write as follows (see [10])

$$M \simeq \frac{s}{h^2k^2} \max_{i,j} \Big| \begin{aligned} & u(x_{i+1},t_{j+1}) - 2u(x_i,t_{j+1}) + u(x_{i-1},t_{j+1}) - \\ & - 2(u(x_{i+1},t_j) - 2u(x_i,t_j) + u(x_{i-1},t_j)) + \\ & + u(x_{i+1},t_{j-1}) - 2u(x_i,t_{j-1}) + u(x_{i-1},t_{j-1}) \Big| \end{aligned} \tag{25}$$

where $i = 1, 2, \ldots, n-1, j = 1, 2, \ldots, m-1$ and $s = 1.5$ (in this case is assumed that the approximation errors in (21) and (22) are not greater than 50%). The terms $u(x_i, t_j)$ for $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, m-1$ are calculated from the conventional central difference method.

Summarizing, the central difference interval method is obtained as follows

$$\begin{aligned} 2\left(V^2\frac{K^2}{H^2} - 1\right) U(X_i,T_j) + U(X_i,T_{j-1}) + U(X_i,T_{j+1}) - \\ -V^2\frac{K^2}{H^2}\left(U(X_{i-1},T_j) + U(X_{i+1},T_j)\right) = E_M, \end{aligned} \tag{26}$$

for $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, m-1$,

where E_M is given by the formula

$$E_M \in \frac{K^2}{12}(H^2 + K^2V^2)[-M, M]. \tag{27}$$

The boundary conditions (2), for $j = 1, 2, \ldots, m$, are as follows

$$\begin{aligned} U(X_0,T_j) &= [0,0], \\ U(X_n,T_j) &= [0,0]. \end{aligned} \tag{28}$$

The initial conditions (3), for $i = 1, 2, \ldots, n-1$, are given as

$$\begin{aligned} & U(X_i,T_0) = \Phi(X_i), \\ & U(X_i,T_1) = \Phi(X_i) + K\Psi(X_i) + \frac{K^2V^2}{2}\Phi''(X_i) + \frac{K^3V^3}{6}\Phi'''(X_i) + E_C, \end{aligned} \tag{29}$$

where E_C is given by the formula

$$E_C \in \frac{K^4V^2}{24}[-M, M]. \tag{30}$$

Value M is calculated by the formula (25).

These relations are satisfied for each mesh point $(x_i, t_j) \in (X_i, T_j)$, where (X_i, T_j) are interval representations (see [11] and [12]) of the suitable mesh points (x_i, t_j). The functions U, Φ, Ψ and the values K, H, V are interval extension of functions u, φ, ψ and values k, h, v respectively.

To find the interval solution at the mesh points of the grid we have to solve the system of $(m-1)\times(n-1)$ interval linear equations. This system we can write in the form

$$\begin{bmatrix} I & & & \\ A & I & & \\ I & A & I & \\ & \ddots & \ddots & \ddots \\ & & I & A & I \end{bmatrix} \cdot \begin{bmatrix} U_2 \\ U_3 \\ \vdots \\ \vdots \\ U_m \end{bmatrix} = \begin{bmatrix} R_2 \\ R_3 \\ \vdots \\ \vdots \\ R_m \end{bmatrix} \tag{31}$$

where I is identity matrix $(n-1)\times(n-1)$, A is tridiagonal matrix in the form

$$A = \begin{bmatrix} 2(\Gamma^2-1) & -\Gamma^2 & & & \\ -\Gamma^2 & \ddots & \ddots & & \\ & \ddots & \ddots & -\Gamma^2 & \\ & & -\Gamma^2 & 2(\Gamma^2-1) \end{bmatrix}_{(n-1)\times(n-1)} \tag{32}$$

where $\Gamma = V \cdot \frac{K}{H}$.

Vectors U_j and R_j, for $j = 2,3,\ldots,m$, are adequately vectors of unknowns and constants and can be presented as follows

$$U_j = [U_{1,j}, U_{2,j}, \ldots, U_{n-1,j}]^T, \quad R_j = [R_{1,j}, R_{2,j}, \ldots, R_{n-1,j}]^T. \tag{33}$$

Elements of vector R_j, for $j = 2,3,\ldots,m$, (together with the Dirichlet (28) and the Cauchy (29)) are obtained by the formulas

$$\begin{aligned}
&i = 1, \quad j = 2: \\
&\qquad R_{i,j} = E_M - 2(\Gamma^2-1)(\Phi(H) + K\Psi(H) + E_C) + \Gamma^2\Phi(jH) + \\
&\qquad\quad + K\Psi(jH) + E_C - \Phi(H), \\
&i = 2,3,\ldots,n-2, \quad j = 2: \\
&\qquad R_{i,j} = E_M + \Gamma^2(\Phi((i-1)H) + K\Psi((i-1)H) + E_C) - \\
&\qquad\quad - 2(\Gamma^2-1)(\Phi(iH) + K\Psi(iH) + E_C) + \\
&\qquad\quad + \Gamma^2(\Phi((i+1)H) + K\Psi((i+1)H) + E_C) - \Phi(iH), \\
&i = n-1, \quad j = 2: \\
&\qquad R_{i,j} = E_M + \Gamma^2(\Phi((i-1)H) + K\Psi((i-1)H) + E_C) - \\
&\qquad\quad - 2(\Gamma^2-1)(\Phi(iH) + K\Psi(iH) + E_C) - \Phi(iH), \\
&i = 1,2,\ldots,n-1, \quad j = 3: \\
&\qquad R_{i,j} = E_M - (\Phi(iH) + K\Psi(iH) + E_C), \\
&i = 1,2,\ldots,n-1, \quad j = 4,5,\ldots,m: \\
&\qquad R_{i,j} = E_M.
\end{aligned} \tag{34}$$

To solve the interval linear systems of equations (31) interval algorithm based on Gaussian elimination with complete pivoting is used.

5 The Floating-Point Interval Arithmetic

The conventional calculations on the computer give solutions with errors. There are initial-data errors, data representation errors, rounding errors and errors of methods. The results are obtained with the use of *IntervalArithmetic* unit written by professor Andrzej Marciniak in the *Delphi Pascal* language. This unit allows to represent all input data in the form of intervals and make all calculations in the floating-point interval arithmetic. Some standard interval functions can be used to obtain the results in the form of intervals. Summarizing, the interval methods for solving PDE with using floating-point interval arithmetic give solutions, in form of intervals, which contain all possible numerical errors.

6 Numerical Experiment

We consider an example of the string (with adequately assumptions), which satisfied equation (1) for $x \in < 0, \pi >$ and $t > 0$.

Three following parameters v are given: $v_1 = 10^{-3}$, $v_2 = 10^{-2}$, $v_3 = 10^{-1}$.

The boundary conditions (2) at the time $t > 0$ are as follows

$$u(0,t) = 0,$$
$$u(\pi,t) = 0,$$

and initial conditions for $t = 0$ and $x \in < 0, \pi >$ are given by

$$u(x,0) = 0$$
$$\left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0} = \sin(x).$$

Table 1. Maximum widths of interval solutions for $v_1 = 10^{-3}, v_2 = 10^{-2}, v_3 = 10^{-1}$

$n = m$ / v	10	20	30	40	50	60	70	80
10^{-3}	$3 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$4 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$8 \cdot 10^{-4}$	$6 \cdot 10^{-4}$
10^{-2}	$3 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$4 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$8 \cdot 10^{-4}$	$6 \cdot 10^{-4}$
10^{-1}	$3 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$4 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$9 \cdot 10^{-4}$	$7 \cdot 10^{-4}$

7 Conclusions

The interval methods of floating point interval arithmetic guarantee correct digits in obtained solutions. The main point is to make good estimations of truncation errors. The presented approximation of an initial condition (with local

truncation error $O(h^4)$) allows to make estimation error together with estimation error of the central difference method for the wave equation. In this case the main point is find one value M (25) which appears in estimations of errors for the wave equation (27) and the initial condition (30).

The larger value v in the wave equation could be considered if the estimations of errors do not depend on velocity v.

Three algorithms for the large system of interval linear equations in the form (31) had been tested of floating point interval arithmetic: the GaussJordan elimination, the Gaussian elimination without and with complete pivoting. The best solutions (which have the minimal widths of interval solutions) are obtained by interval method based on Gaussian elimination with complete pivoting.

References

1. 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)
2. Gajda, K., Marciniak, A., Szyszka, B.: Three- an Four-Stage Implicit Interval Methods of Runge-Kutta Type. Computational Methods in Science and Technology 6, 41–59 (2000)
3. Marciniak, A., Szyszka, B.: One- and Two-Stage Implicit Interval Methods of Runge-Kutta Type. Computational Methods in Science and Technology 5, 53–65 (1999)
4. Marciniak, A., Szyszka, B.: On Representation of Coefficients in Implicit Interval Methods of Runge-Kutta Type. Computational Methods in Science and Technology 10(1), 57–71 (2004)
5. Szyszka, B., Marciniak, A.: An interval method for solving some partial differential equations of second order. In: 80th Annual Meeting of the International Association of Applied Mathematics and Mechanics, GAMM, Proceedings on CD (2009)
6. Szyszka, B.: Central difference interval method for solving the wave equation. In: Simos, T.E., Psihoyios, G., Tsitouras, C. (eds.) ICNAAM 2010. AIP Conference Proceedings, vol. 1281, pp. 2173–2176 (2010)
7. Evans, L.C.: Partial Differential Equations. PWN Warszawa (2008) (in Polish)
8. Kacki, E.: Partial Differential Equations in physics and techniques problems. WNT Warszawa (1995) (in Polish)
9. Burden, R.L., Faires, J.D.: Numerical Analysis. Prindle, Weber and Schmidt, Boston (1985)
10. Marciniak, A.: An Interval Difference Method for Solving the Poisson Equation the First Approach. Pro. Dialog. 24, 49–61 (2008)
11. Kalmykov, S.A., Schokin, Y.I., Yuldaschew, Z.H.: Methods of Interval Analysis. Nauka, Novosibirsk (1986)
12. Moore, R.E.: Methods and Applications of Interval Analysis. SIAM (1979)

Meta-model Assisted Evolutionary Optimization of Cellular Automata: An Application to the SCIARA Model

Donato D'Ambrosio[2], Rocco Rongo[1], William Spataro[2], and Giuseppe A. Trunfio[3]

[1] Department of Earth Sciences, University of Calabria, 87036 Rende (CS), Italy
[2] Department of Mathematics, University of Calabria, 87036 Rende (CS), Italy
[3] DADU, University of Sassari, 07041 Alghero (SS), Italy

Abstract. The automatic optimization of Cellular Automata (CA) models often requires a large number of time-consuming simulations before an acceptable solution can be found. As a result, CA optimization processes may involve significant computational resources. In this paper we investigate the possibility of speeding up a CA calibration through the approach of meta-model assisted search, which is widely used in many fields. The adopted technique relies on inexpensive surrogate functions able to approximate the fitness corresponding to the CA simulations. The calibration exercise presented here refers to SCIARA, a CA for the simulation of lava flows. According to the preliminary results, the use of meta-models enables to achieve a significant gain in computational time.

Keywords: Cellular Automata, Model Calibration, Meta-modelling.

1 Introduction

In many fields, Cellular Automata (CA) models are considered a valuable tool for simulating the dynamic evolution, over time and space, of systems whose behaviour emerges from local interactions. In most CA models of real complex systems: (*i*) each cell corresponds to a portion of a real space (e.g. a location on the Earth's surface); (*ii*) the states of the cell correspond to spatial characteristics which are important to the model (e.g. slope, temperature); (*iii*) the transition function models some local interactions among the system components; (*iv*) each CA step corresponds to an interval of time. In addition, the transition function is often dependent on some scalar parameters (e.g. [1,2]).

The reliability of such parameterized CA models is maximized by some standard procedures, such as *model calibration*. The calibration process consists of finding the values of the model parameters that are not exactly known and cannot be directly measured, in such a way that the model itself better mimics the observed dynamic of the system under consideration. Since the search space is usually large, automated methods have been developed by defining calibration as an optimization problem in which the solution in terms of parameter values must

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 533–542, 2012.

maximise an objective function (i.e. the fitness measure). At present, many optimization algorithms have been exploited for the automatic calibration of CAs, ranging from techniques based on exhaustive exploration of the search space to the use of heuristics such as Evolutionary Algorithms (EA) [2,3]. Usually, such a search process requires a large number of fitness evaluations (i.e. CA simulations) before an acceptable solution can be found. Unfortunately, carrying-out CA simulations of real complex systems is often computationally expensive. A typical way to deal with the resulting cost of the optimization process is the use of parallel computing [2]. An additional strategy can be based on inexpensive surrogate functions able to approximate the fitness corresponding to the CA simulations. Such an approach, known as meta-model assisted (or surrogate assisted) optimization, has been widely applied in engineering design but, to our knowledge, its advantages in calibrating CA have not yet been tested. In the present study this opportunity is explored with reference to the calibration of SCIARA [4,5], which is a CA for the simulation of lava flows. In particular, the paper presents some preliminary results of an empirical investigation aimed at highlighting in CA calibration the benefits of some typical meta-models, namely the Artificial Neural Network (ANN), the General Regression Neural Network (GRNN) and a simple second order polynomial approximation.

The paper is organised as follows. In section 2 the CA calibration problem is formalized. Section 3 outlines the tested meta-modelling approaches within an evolutionary search. Section 4 illustrates the results of the numerical experiments and section 5 concludes the paper outlining possible future work.

2 Optimization of Cellular Automata

In many applications of the CA modelling approach the cells' transition function depends on a vector of constant parameters $\mathbf{p} = [p_1, \ldots, p_n]^T$, which belongs to a space Λ (e.g. [1,2]). In particular, the overall transition function Φ gives the global configuration $\Omega^{(t+1)}$ (i.e. the set of all cell states) at the step $t+1$ as:

$$\Omega^{(t+1)} = \Phi(\Omega^{(t)},\ \mathbf{p}) \tag{1}$$

The iterative application of Φ, starting from an initial configuration $\Omega^{(0)}$, leads to the CA simulation:

$$\Omega^{(0)} \xrightarrow{\Phi} \Omega^{(1)} \xrightarrow{\Phi} \cdots \xrightarrow{\Phi} \Omega^{(t)} \quad \Longrightarrow \quad \Omega^{(t)} = \Phi^t(\Omega^{(0)}, \mathbf{p}) \tag{2}$$

where the dependence of the automaton configuration at the time step t on both the initial configuration and the parameters is explicit, with the other automaton characteristics (i.e. the model structure) being fixed.

The model can be optimized with respect to $\mathbf{p}$ to maximise the agreement between the simulated patterns and the real ones. To formalize the problem, let us suppose the existence of a spatio-temporal dataset $\bar{\mathcal{V}}$ collecting some automaton configurations, which come from an experiment of the real system behaviour. In particular, let $\bar{\mathcal{V}}$ be composed by a sequence of q configurations:

$$\bar{\mathcal{V}} = \left\{ \bar{\Omega}^{(k)} : k \in \{0, \tau_1, \ldots, \tau_q\} \right\} \tag{3}$$

where τ_i indicates the instant of time in which the configuration $\bar{\Omega}^{(i)}$ is known. Starting from $\bar{\Omega}^{(0)}$, and given a vector $\mathbf{p}$ of parameters, the process (2) can be executed for the computation of the $q-1$ configurations:

$$\mathcal{V} = \left\{ \Omega^{(k)} : k \in \{\tau_1, \ldots, \tau_q\} \right\} \tag{4}$$

where $\Omega^{(j)} = \Phi^j(\bar{\Omega}^{(0)}, \mathbf{p})$. The agreement θ between the real and simulated processes is usually measured by a suitable fitness function:

$$\theta = \Theta\left(\bar{\mathcal{V}}, \mathcal{V}\right) = \Theta\left(\bar{\mathcal{V}}, \mathbf{p}\right) \tag{5}$$

Therefore, the calibration consists of the following maximisation problem:

$$\max_{\mathbf{p} \in \Lambda} \Theta\left(\bar{\mathcal{V}}, \mathcal{V}\right) \tag{6}$$

which involves finding a proper value of $\mathbf{p}$ that leads to the best agreement between the real and simulated spatio-temporal sequences.

Given the form of the fitness function, different heuristics and in particular EAs have been used to tackle the automatic solution of problem (6) [2,3]. In this paper a Genetic Algorithm (GA) [13] assisted by different types of fitness approximation models was adopted. The GA is used to evolve a randomly initialized population, whose generic *chromosome* is a vector encoding an n-dimensional vector of parameters $\mathbf{p}$. The i-th element of the chromosome is obtained as the binary encoding of the parameter p_i, using a suitable number of bits and given its set of definition. Each chromosome can be decoded in a vector of parameters $\mathbf{p}$ and, through performing a CA simulation, the function Θ can be computed.

3 Metamodel-Assisted Evolutionary Optimization

In the meta-model assisted GA, the original fitness evaluations are partly replaced by the fitness estimates provided by less expensive models. This allows to reduce the number of CA simulations needed to evaluate the individuals that are generated by the standard genetic operators during the evolutionary search. In the meta-model assisted evolutionary optimization, two main approaches exist for building the surrogate fitness function. In particular, in the so-called *off-line learning* the model is built before the optimization starts exploiting existing data (e.g. patterns available from previous optimization runs). A different strategy, named *online learning*, consists of constructing the surrogate fitness model using data that are generated during the optimization process. Since the latter approach has been reported to be more successful and reliable than the former [6], it has been adopted in this study.

In the present application, the CA simulations carried out during the evolutionary search provide a training set $\mathcal{T}_s$:

$$\mathcal{T}_s = \left\{ \langle \theta^{(1)}, \mathbf{p}^{(1)} \rangle, \langle \theta^{(2)}, \mathbf{p}^{(2)} \rangle, \ldots, \langle \theta^{(n_t)}, \mathbf{p}^{(n_t)} \rangle \right\} \tag{7}$$

where each fitness value $\theta^{(i)}$ corresponds to a parameter vector $\mathbf{p}^{(i)}$. Thus, on the basis of the patterns in $\mathcal{T}_s$ a meta-model $\hat{\Theta}$ can be dynamically built for evaluating the candidate solutions generated by the EA, that is for the estimation of the unknown fitness value corresponding to a vector $\mathbf{p}$:

$$\hat{\theta} = \hat{\Theta}(\mathbf{p}) \tag{8}$$

In the case of online model building, either a global meta-model or a local one can be used [7]. However, often an accurate global approximation of the true fitness function is difficult to obtain. This is particularly true when dealing with high-dimensional and multimodal fitness landscapes [7]. In these cases, local meta-models can be constructed using a suitable subset of the training patterns in $\mathcal{T}_s$. For example, a surrogate of the real fitness can be constructed for each individual $\mathbf{p}$ to be evaluated using only the k nearest neighbours of $\mathbf{p}$ in $\mathcal{T}_s$. As an alternative, the set $\mathcal{T}_s$ can be partitioned in subsets (e.g. using the k-means clustering) and a local meta-model can be constructed for each of them. Clearly, a local meta-model can potentially be more accurate. However, the cost of training a number of local surrogates should be compared with the cost of the true fitness evaluation. Nevertheless, a CA simulation is often much more expensive than building a local surrogate.

As mentioned above, the use of meta-models in place of CA simulations may significantly reduce the overall computational cost of the CA optimization process. However, an optimum for the meta-model not always is also an optimum for the true fitness function. Therefore, to avoid convergence to false optima, the surrogate-assisted optimization should also use, in some way, the true fitness function. On the other hand, the involvement of the latter should be minimized due to its high computational cost. A trade-off is provided by a suitable *evolution control* strategy [7]. In particular, two different approaches are commonly adopted. The first is the individual-based control, which was adopted in this paper and consists of using the true fitness function to evaluate some of the individuals produced at each generation (i.e. the *controlled individuals*). The second is a generation-based control, in which all the individuals of some generations (i.e. the *controlled generations*) are evaluated through the true fitness function. In the case of individual-based control, different strategies can be used to choose the controlled individuals. In this paper the so-called *best strategy* was adopted, which consists of assigning their exact fitness to some of the individuals that are the best according to the meta-model.

Below, the different meta-modelling techniques that were tested for the optimization of SCIARA are briefly described.

3.1 Feed Forward ANN

The use of ANNs is one of the most common approaches in meta-model assisted EAs [7]. In this study, a feedforward multilayer perceptron (MLP) has been used

as a SCIARA surrogate during the calibration process. The MLP architecture was determined comparing many different fully connected networks on a training set obtained from previous SCIARA runs. The adopted MLP has one input layer with 5 units (as explained below in more detail, SCIARA has 5 parameters to calibrate), two hidden layers with 7 and 2 units respectively, and one output neuron. Formally, it can be described by the following equation:

$$\bar{\theta}(\mathbf{p}) = f\left(\sum_{j=1}^{h^{(2)}} w_j\, f\left(\sum_{k=1}^{h^{(1)}} w_{kj}^{(2)}\, f\left(\sum_{i=1}^{n} w_{ik}^{(1)}\, \bar{p}_i + w_{0k}\right) + w_{0j}\right) + w_0\right) \tag{9}$$

where $\bar{\theta}$ is the normalized CA fitness estimate, $f(x) = 1/(1+e^{-x})$ is the logistic activation function, $h^{(1)}$ and $h^{(2)}$ are the number of hidden units, $\bar{p}_i$ are the normalized CA parameters and w are the unknown weights to be learned. The resilient back-propagation (RProp) algorithm [8] was used for training the network. In particular, to improve the generalization ability of the MLP an early stopping approach [9] was adopted by splitting the set $\mathcal{T}_s$ into a training set and a validation set.

3.2 General Regression Neural Network

The name GRNN indicates the Nadaraya-Watson Kernel Regression Estimator [10,11]. As shown by Specht in [12], the method can be interpreted as an ANN. Adopting the most used Gaussian kernel, the GRNN gives an estimation of the fitness $\theta(\mathbf{p})$ corresponding to the CA parameter vector $\mathbf{p}$ as:

$$\theta(\mathbf{p}) = \frac{\displaystyle\sum_{i=1}^{n_t} \theta^{(i)}\ \exp\left(-\sum_{j=1}^{n} \frac{|\, p_j - p_j^{(i)}\,|^2}{2\sigma_j^2}\right)}{\displaystyle\sum_{i=1}^{n_t} \exp\left(-\sum_{j=1}^{n} \frac{|\, p_j - p_j^{(i)}\,|^2}{2\sigma_j^2}\right)} \tag{10}$$

where n_t is the number of training points in $\mathcal{T}_s$, $\theta^{(i)}$ is the i-th fitness value obtained by a previous CA simulation based on the parameter vector $\mathbf{p}^{(i)}$. In Equation (10) the so-called *bandwidths* σ_i are positive numbers to be estimated on the basis of the training data. A high bandwidth corresponds to a low sensitivity of the estimated fitness to the distance from the sampling points. Hence, the use of bandwidths larger than optimal leads to over-smoothing of data. Conversely, smaller than optimal values of σ_i produce overfitting of data. In the present application the σ_i (one value for each CA parameter to optimize) were determined using the conjugate gradient (CG) method to minimize the mean squared error (MSE) of the model. The latter was computed according to the k-fold cross-validation approach in which the training data set $\mathcal{T}_s$ is partitioned into k subsets: one of them is used as a validation data set to calculate the MSE, and the remaining $k-1$ subsets are used as training data. The cross-validation

process is repeated k times (i.e. the number of the so-called *folds*), with each of the k subsets used exactly once as validation data. The k MSEs from all folds are then averaged to produce a single MSE.

3.3 Polynomial Approximation

The adopted polynomial surrogate is the following Quadratic Polynomial Approximation (QPA):

$$\theta(\mathbf{p}) = \beta_0 + \sum_{1 \le i \le n} \beta_i \, p_i + \sum_{1 \le i \le j \le n} \beta_{(n-1+i+j)} \, p_i \, p_j = \boldsymbol{\beta}^T \, \tilde{\mathbf{p}} \tag{11}$$

where:

$$\boldsymbol{\beta} = [\beta_0, \, \beta_1, \, \ldots, \, \beta_{n_v - 1}]^T \tag{12}$$

is the vector collecting the $n_v = (n+1)(n+2)/2$ model coefficients and:

$$\tilde{\mathbf{p}} = [1, \, p_1, \, p_2, \, \ldots, \, p_1 p_2, \, \ldots, \, p_n^2]^T \tag{13}$$

is the vector of the CA parameters mapped into the polynomial model. In this study, the model coefficients $\boldsymbol{\beta}$ are estimated using the least square method.

In particular, an ad-hoc local quadratic meta-model is built for each individual $\mathbf{p}$ on the basis of its $n_s \ge n_v$ nearest neighbours in $\mathcal{T}_t$.

4 Calibration Benchmark Tests and Discussion

The approach outlined above has been applied to the last release of SCIARA, a CA model for lava flows simulation. The model, which is described in detail in [5], is based on a square grid and accounts for the relevant physical processes involved in the macroscopic phenomenon. In brief, a specific component of the transition function computes lava outflows from the central cell towards its neighbouring ones on the basis of the altitudes, lava thickness and temperatures in the neighbourhood. The model is based on a Bingham-like rheology and therefore the concepts of critical height and viscosity are explicitly considered. In particular, lava can flow out if and only if its thickness overcomes a critical value (critical height), so that the basal stress exceeds the yield strength. The critical height mainly depends on the lava temperature according to a power law. Moreover, viscosity is accounted in terms of flow relaxation rate, being this latter the parameter of the distribution algorithm that influences the amount of lava that actually leaves the cell. Two mechanisms determine at each time-step the new cell temperature. The first one takes into account the mass and energy exchange between neighbouring cells, while the second updates the temperature by considering thermal energy loss due to lava surface irradiation. The temperature variation, besides the change of critical height, may lead to the lava solidification which, in turn, determines a change in the morphology. In SCIARA the transition function depends on some scalar parameters, invariant in time and space. Among these are: r_s, the relaxation rate at the temperature of solidification;

Table 1. Parameters object of calibration, explored ranges and target values

Parameter	Explored range for calibration	Target value
r_s	[0, 1]	0.096
r_v	[0, 1]	0.853
h_s $[m]$	[1, 50]	13.67
h_v $[m]$	[1, 50]	1.920
p_c	[0, 100]	8.460

r_v, the relaxation rate at the temperature of extrusion; h_s, the critical height at the temperature of solidification; h_v, the critical height at the temperature of extrusion; p_c, the "cooling parameter", which regulates the thermal energy loss due to lava surface irradiation. Once that the input to the model has been provided, such as parameter values, orography, vents and the effusion rates as a function of time, SCIARA can simulate the lava flow. The simulation stops when the fluxes fall below a small threshold value. However, before using the model for predictive applications, the parameters must be optimized for a specific area and lava type. To this end, the following fitness measure was defined:

$$\theta = \frac{m(R \cap S)}{m(R \cup S)} \tag{14}$$

where R and S represent the areas affected by the real and simulated event, respectively, while $m(A)$ denotes the measure of the set A. Note that $\theta \in$[0,1]; its value is 0 if the real and simulated events are completely disjoint, being $m(R \cap S)$=0; it is 1 in case of perfect overlap, being $m(R \cap S) = m(R \cup S)$.

For the calibration task the meta-models mentioned above were used to support a standard genetic algorithm (SGA) [13]. In the latter, a population of bit-strings, each encoding a candidate solution $\mathbf{p} = [r_s, r_v, h_s, h_v, p_c]$, is evolved through the iterative application of the operators of selection, recombination and mutation. In particular, each of the SCIARA parameters was encoded on a string of 10 bits using the intervals shown in Table 1. The SGA used the standard 1-point crossover applied with probability $p_c = 0.8$, while the mutation consisted of a bit flipping with a fixed probability per bit defined as $p_m = 1/n_b$, being n_b the number of bits composing the individual. The standard Roulette Wheel Selection [13] was applied together with an elitist replacement scheme, that is the best individual in the population was always preserved from generation to generation. Moreover, all the GAs operated on a population of 50 individuals.

In the calibration exercise discussed here the different heuristics were applied on a real event occurred on Mt. Etna (Sicily, Italy) in 2001 which is described in details in [4]. However, the target final configuration was obtained with SCIARA itself, using the set of parameters shown in Table 1. This guarantees the existence of a zero-error solution of the calibration problem, thus allowing for an unbiased evaluation of the calibration procedures. In Figure 1 two examples of landscapes generated by the fitness defined in Equation (14) are depicted. In particular, each

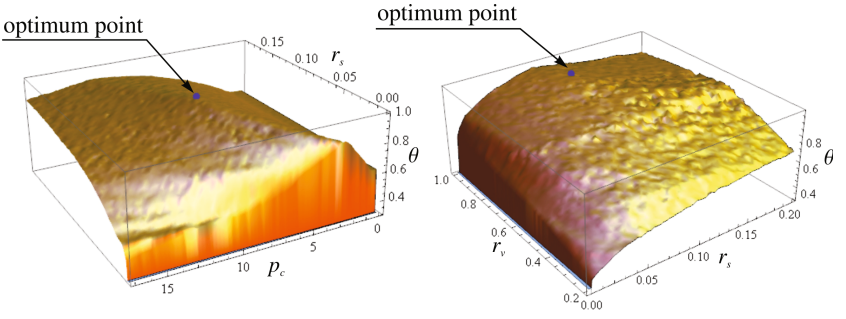


Fig. 1. Example of rugged fitness landscapes generated by SCIARA. The surfaces refer to a neighbourhood of the target point shown in Table 1.

Table 2. Overview of the calibration results obtained assigning to each search algorithm a budget of 1000 SCIARA evaluations. The statistics were computed on 10 independent run of each algorithm.

	π	Average	Min	Max	Std. Dev.
SGA	-	0.816	0.738	0.888	0.007
MLP-GA	0.1	0.886	0.702	0.961	0.105
	0.2	0.924	0.868	0.957	0.036
	0.3	0.907	0.774	0.956	0.076
GRNN-GA	0.1	0.924	0.908	0.955	0.018
	0.2	0.910	0.819	0.964	0.057
	0.3	0.881	0.812	0.931	0.045
QPA-GA	0.1	0.917	0.843	0.952	0.043
	0.2	0.889	0.816	0.920	0.044
	0.3	0.908	0.874	0.955	0.032

of the two surfaces was obtained executing about 1000 SCIARA simulations in a neighbourhood of the target point shown in Table 1. As can be seen, the fitness landscape is significantly rugged. This tends to slow down the convergence of optimization heuristics and makes the case of SCIARA particularly suited to test the benefits of the meta-model assisted optimization. In fact, as found in previous studies [14], one of the advantages of surrogate models lies in their capability of smoothing irregular fitness landscapes to prevent the search from getting stuck in local optima. As mentioned above, the meta-model assisted GAs were based on an individual-based control strategy. In particular, at the first generation all the individuals were evaluated using a SCIARA simulation. Then, at the subsequent generations: (*i*) all the offspring individuals were pre-evaluated using the surrogate model; (*ii*) a fixed fraction π of the best offsprings (according to the meta-model) was re-evaluated using a SCIARA simulation. In particular, three different value of π were considered, namely 0.1, 0.2 and 0.3. Besides the SGA, three types of surrogate-assisted versions were tested: (*i*) the MLP-GA, based on the MLP described in section 3.1; (*ii*) the GRNN-GA, based on the GRNN described in section

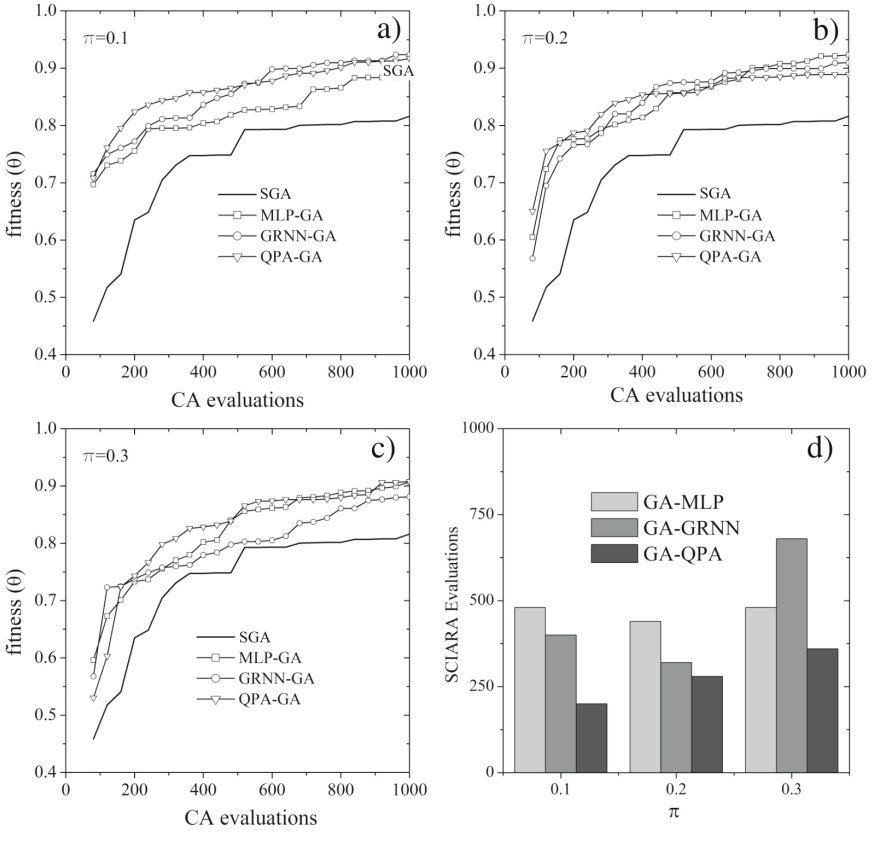


Fig. 2. Charts a), b) and c) refer to different value of the ratio π of controlled individuals at each generation. Figure d) shows the number of SCIARA simulations that were required to achieve the same fitness obtained by the SGA with 1000 simulations.

3.2; (*iii*) the QPA-GA, based on the QPA model described in section 3.3. For each type of heuristic search, 10 independent runs were carried out. As shown in Table 2, the SGA achieved an average fitness value of $\theta = 0.816$ and a maximum of $\theta = 0.888$. All the meta-model assisted GAs were able to outperform the SGA. In particular, all the heuristics under comparison provided, for different values of π, a final average $\theta \approx 0.92$ and a maximum $\theta \approx 0.96$. It is important to remark that, because of its nature, a small gain in the fitness defined by Equation (14) corresponds to a significant difference in the final map of the lava invasion. Figures 2-*a*, *b* and *c* show the average behaviour of the meta-model assisted algorithms during the search process, together with the behaviour of the SGA. Interestingly, for any number of SCIARA simulations the surrogate-assisted heuristics attained an average fitness significantly higher than that of the SGA. This can also be observed from the different point of view highlighted in Figure 2-*d*. The latter shows the number of simulations that were needed to the assisted heuristics to achieve, on average, the same fitness given by the SGA after 1000 simulations.

5 Conclusions and Future Work

The preliminary results of this study indicate that the automatic optimization of CA models can greatly benefit by the use of meta-models. Nevertheless, there may be significant potential for further improvements. In particular, a promising direction to explore is that of local search. Once there is a surrogate in analytical form, it is possible to locally improve an individual using classical gradient-based methods. Eventually, the locally optimized vector of parameters can be encoded back into the population according to a Lamarckian evolutionary approach.

References

1. Di Gregorio, S., Serra, R.: An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. Future Generation Computer Systems 16, 259–271 (1999)
2. Spataro, W., D'Ambrosio, D., Rongo, R., Trunfio, G.A.: An Evolutionary Approach for Modelling Lava Flows Through Cellular Automata. In: Sloot, P.M.A., Chopard, B., Hoekstra, A.G. (eds.) ACRI 2004. LNCS, vol. 3305, pp. 725–734. Springer, Heidelberg (2004)
3. Blecic, I., Cecchini, A., Trunfio, G.A.: A Comparison of Evolutionary Algorithms for Automatic Calibration of Constrained Cellular Automata. In: Taniar, D., Gervasi, O., Murgante, B., Pardede, E., Apduhan, B.O. (eds.) ICCSA 2010, Part I. LNCS, vol. 6016, pp. 166–181. Springer, Heidelberg (2010)
4. Crisci, G.M., Rongo, R., Gregorio, S.D., Spataro, W.: The simulation model SCIARA: the 1991 and 2001 lava flows at Mount Etna. Journal of Volcanology and Geothermal Research 132, 253–267 (2004)
5. Spataro, W., Avolio, M.V., Lupiano, V., Trunfio, G.A., Rongo, R., D'Ambrosio, D.: The latest release of the lava flows simulation model sciara: First application to Mt Etna (Italy) and solution of the anisotropic flow direction problem on an ideal surface. Procedia CS 1, 17–26 (2010)
6. Willmes, L., Bäck, T., Jin, Y., Sendhoff, B.: Comparing neural networks and kriging for fitness approximation in evolutionary optimization. In: IEEE Congress on Evolutionary Computation, vol. (1), pp. 663–670 (2003)
7. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput 9, 3–12 (2005)
8. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: IEEE International Conference on Neural Networks, pp. 586–591 (1993)
9. Morgan, N., Bourlard, H.: Advances in neural information processing systems 2, pp. 630–637. Morgan Kaufmann Publishers Inc., San Francisco (1990)
10. Nadaraya, E.: On estimating regression. Theory of Probability and Its Applications 9 (1964)
11. Watson, G.: Smooth regression analysis. Sankhya Series A, 359–372 (1969)
12. Specht, D.F.: A general regression neural network. IEEE Transactions on Neural Networks 2, 568–576 (1991)
13. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, 1st edn. Addison-Wesley Professional (1989)
14. Liang, K.H., Yao, X., Newton, C.: Evolutionary search of approximated n-dimensional landscapes. International Journal of Knowledge-based Intelligent Engineering Systems 4, 172–183 (2000)

How the Competitive Altruism Leads to Bistable Homogeneous States of Cooperation or Defection

Andrzej Jarynowski[1], Przemysław Gawroński[2], and Krzysztof Kułakowski[2]

[1] Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, ul. Reymonta 4, PL-30059 Kraków, Poland
[2] Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, PL-30059 Kraków, Poland
andrzej.jarynowski@uj.edu.pl, gawron@newton.ftj.agh.edu.pl, kulakowski@novell.ftj.agh.edu.pl

Abstract. Our recent minimal model of cooperation (K. Kulakowski and P. Gawronski, Physica A 388 (2009) 3581) is modified as to allow for time-dependent altruism. This evolution is based on reputation of other agents, which in turn depends on history. We show that this modification leads to two absorbing states of the whole system, where the cooperation flourishes in one state and is absent in another one. The effect is compared with the results obtained with the model of indirect reciprocity, where the altruism of agents is constant.

Keywords: cooperation, altruism, reputation, Prisoner's Dilemma, reciprocity, simulation.

1 Introduction

The Prisoner's Dilemma (PD) is a canonical example of a game where mutual cooperation is not profitable for an individual player and simultaneously it is profitable for a society. This paradoxical aspect and the wide set of situations where PD applies makes it a central point in game theory. On the sociological side, PD is considered as one of three generic games which represent three paradigmatic interaction situations; coordination and inequality are two others [1]. As a rule, PD is investigated in the frames of game theory, i.e. decisions are made on the basis of calculated payoffs. However, many scholars indicated that these frames are too narrow [1,2,3,4]. Here we are interested in a model of possible cooperation where both aspects are captured, the normative one and the rational one. Our starting point is the social mechanism of competitive altruism [5,6], where individuals compete for most altruistic partners. This mechanism was the basis of our previous minimal model of cooperation, where players differed in altruism; the latter was defined as a willingness to cooperate [7]. The agents' behaviour was encoded in the form of their time-dependent reputation.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 543–550, 2012.

Our nice result was that altruistic players were rewarded by cooperation of other agents. In this way, the classical deficiency of altruistic strategy - being abused by selfish free-riders - was evaded in our model.

The aim of this paper is to build into the model the fact that people learn, and their willingness to cooperate is modified by their personal experience. The dynamics of one's willingness to cooperate is at the core of the phenomenon of competitive altruism. It is clear that in competing with others, one must modify own behaviour as to outperform the rivals. Therefore, individual altruisms should vary as well, if the second premise is to be included to the model. Having this in mind, we propose here two schemes of varying the players' altruism after each game. According to our first option, say A, altruism is updated in the same way as the reputation in [7], only somewhat slower. This small difference seems realistic: we change our opinions on other people almost immediately after getting new information about them. Also, the difference is dictated by our aim to compare the results with those in [7], where altruism of each agent was constant. Second option B introduces a modification which is suggested by Scheff theory of shame and pride [8,9]. According to this theory, a mutual respect of two agents expressed by their cooperation enhances their self-evaluation, what in turn reinforces their willingness to cooperate. On the other hand, a cooperating agent is humiliated when meets a defection, what reduces her/his willingness to cooperate.

How these modifications of altruism influence the ability of the population to cooperate? In the option A, strategies of cooperation and of defection are equivalent, then the solution should be symmetric with respect to interchange of these strategies. As we explain in detail in the next section, in the option B two succesful cooperators have their altruisms increased, but this event can happen with the probability equal to, roughly, squared concentration of cooperators. If this concentration is initially $1/2$, the process should be neutralized by a decrease of altruism of an unsuccessful cooperator. Below we show that this is not the case; the rules, apparently symmetric, promote the cooperation. Actually, it seems that the coupling between the willingnesses to cooperate of different players is more general than the effect of competitive altruism or of the loops of shame and pride. This coupling can be viewed as a general ability of a set of players to establish a social norm of cooperation, and this norm in turn allows to define a social group. The role of norms in establishing social groups and societies is much too wide a subject to be discussed here [3,10]. Instead, we refer only to the definition of a social norm [11], which clearly underlines the role of mutual expectation of agents: once they believe they recognize the attitudes of the others, a germ of a norm is established.

This ambiguity suggests, that it is desirable to look for another mechanism, not due to the modification of altruism, which leads to an enhancement of cooperation. Our choice is to check the scenario where altruism of each agent is constant, but the modification of reputation depends on the reputation of coplayer. In this way, the parameter controlling the change of reputation of agent i is just the reputation of i's coplayer j. This choice, marked here as option C, is motivated by the mechanism of indirect reciprocity and punishment, discussed,

e.g., in [12]. In other words, the modification of reputation in a game with an agent with bad reputation remains small.

The outline of the paper is as follows. In the next section we explain the original version of the model [7] and options A, B and C considered here. In the third section numerical results are described. There we demonstrate that when altruism is varied, all players defect or all cooperate in the stationary state. This bistability with two homogeneous states is the main result of the paper. We show also, that for the option C, cooperation is promoted.

2 The Model

The system is equivalent to a fully connected graph of N nodes, with a link between each pair of agents at the nodes. Each agent i is endowed with two parameters: altruism ε_i and reputation W_i. Initial values of the parameters are selected randomly from homogeneous distributions: $\rho(\varepsilon_i)$ is constant for $E_1 < \varepsilon_i < E_2$, otherwise $\rho(\varepsilon_i) = 0$, and $\rho(W_i)$ is constant for $V_1 < W_i < V_2$, otherwise $\rho(W_i) = 0$.

During the simulation, a pair of nodes (i, j) is selected randomly. The probability that i cooperates with j is

$$P(i, j) = F(\varepsilon_i + W_j) \tag{1}$$

where $F(x) = 0$ if $x < 0$, $F(x) = x$ if $0 < x < 1$ and $F(x) = 1$ if $x > 1$. In [7], it was only reputations W_i, W_j what evolved in time. If i cooperated, her/his reputation was transformed as $W_i \to (1 + W_i)/2$, otherwise $W_i \to W_i/2$. Here we set W_i to change in the same way.

Moreover - and this is a new element - we allow also the altruism to change. In the option A, this change is ruled according to a similar prescription. If i, j play, the altruism of j varies as

$$\varepsilon_j \to \varepsilon_j + (\pm 1/2 - \varepsilon_j)x \tag{2}$$

where $0 < x < 1$ is a parameter which measures the velocity of change of altruism, the sign $+1$ applies if i cooperates and the sign -1 - if i defects. In other words, the altruism ε_j increases if j meets cooperation, decreases otherwise. As long as $x < 1/2$, the time evolution of altruism is slower than the one of reputation. Here we set $x = 0.1$. Larger values of x just speed up the changes of agents' altruism.

In the option B, it is only the rule of variation of ε_i what is changed with respect of the option A. Namely, if both i and j cooperate, their altruism increase as

$$\varepsilon_i \to \varepsilon_i + (1/2 - \varepsilon_i)x \tag{3}$$

$$\varepsilon_j \to \varepsilon_j + (1/2 - \varepsilon_j)x \tag{4}$$

If i cooperates and j defects, then the altruism of i is reduced as

$$\varepsilon_i \to \varepsilon_i + (-1/2 - \varepsilon_i)x \tag{5}$$

whilst ε_j is not changed. When both i and j defect, nothing is changed.

In the option C, altruism remains constant, as in [7]. The velocity of the variation of reputation of i is controlled by the reputation of her/his coplayer j. Namely, when i cooperates, then her/his reputation W_i changes as

$$W_i \to W_i(1 - zW_j) + zW_j \tag{6}$$

where z is a parameter, which controls the speed of this change. When i defects,

$$W_i \to W_i(1 - zW_j). \tag{7}$$

3 Results

For the option A, the problem is symmetric with respect to an interchange of the strategies: cooperation and defection. When also the initial distributions of both reputation and altruism are symmetric ($V_1 + V_2 = 1$ and $E_1 + E_2 = 0$), this symmetry should be preserved also in the solution. This is so, however, only in the statistical sense. For each simulation, the system breaks the symmetry spontaneously and the time evolution leads to one of two homogeneous states, where each agent adopts the same strategy. In one of these two states, all agents cooperate; in another, all defect. The process of spontaneous symmetry breaking is visible in Fig. 1. There, we show the probability distributions of reputation and altruism for $N = 10^3$ agents at the early stage of the process, after 10^4 games. For the symmetric initial state where $\bar{\varepsilon} = 0$ and $V_1 = V_2 = 0.5$, the result obtained numerically as an average over 10^3 systems, each after 10^5 games, is that the probability of the state "all cooperate" is 0.48. As a rule, this probability is found to be close to 0.5 at the straight line $\bar{W} = 1/2 - \bar{\varepsilon}$. Above this line all cooperate, below this line all defect, except the vicinity of the line (of width about 0.1 for $N = 10^3$), where the probability of the state "all cooperate" changes continuously from 0 to 1. This means that a manipulation of the initial state $(\bar{\varepsilon}, \bar{W})$ modifies the final result, which still remains homogeneous. In particular, when the initial value of $\bar{\varepsilon}$ is shifted by 0.1 downwards, all defect; upwards, all cooperate. The properties of the boundary $\bar{W} = 1/2 - \bar{\varepsilon}$ are a consequence of the adopted form of $P(i, j)$, but the homogeneity of stationary states comes from the system dynamics. The results are obtained for $x = 0.1$; an increase of x just speeds the process up. For $x = 1/2$, the time dependent mean square root of reputation and altruism remain equal, when decreasing to zero.

For the option B, we observe the same homogeneous states "all cooperate" or "all defect", but the probabilities of these states are different. Again we use the same initial reputation for all agents, and the same statistics. For each system of $N = 10^3$ agents we made three runs, each of 100 timesteps, with different initial conditions: *i)* $V_1 = -0.5$ and $V_2 = 0.3$, *ii)* $V_1 = -0.4$ and $V_2 = 0.4$, *iii)* $V_1 = -0.3$ and $V_2 = 0.5$. While the initial conditions *ii)* are neutral, the case *i)*

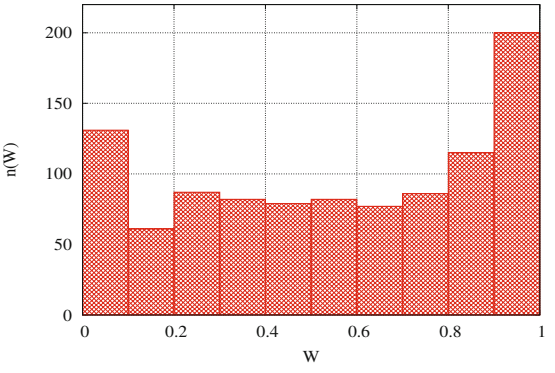


Fig. 1. The reputation distribution $n(W)$ for an exemplary system of $N = 10^3$ agents, evolving according to the option A, at the early stage after 10^4 games. Here, the initial distribution of altruism is symmetric (case *ii)*). After a relatively quick development of groups of agents with good and bad reputation, slowly those of good reputation start to dominate - the right maximum grows larger.

promotes defection and the case *iii)* promotes cooperation. For the option A, the same initial conditions served to demonstrate the symmetry of two strategies. However, for the option B we observe that in the case *ii)* cooperation prevails. The obtained numbers for *i)* are $W_i = 0$ for all agents, and the average ε_i for each system is -0.19 ± 0.15. In the case *iii)* all cooperate, and their altruism is $+0.5$. In the case *ii)* we get again two homogeneous states: "all cooperate" with probability 0.88, "all defect" with probability 0.12. In the former state, the altruism of all agents is maximal: 0.5. In the latter, for each system the average ε is $-0.15 \pm .11$. Note that in the option B, the altruism of uncooperative agents remains unchanged, hence its spread in the uncooperative phase. An exemplary plot of the altruism in this case is shown in Fig. 2. Again, these results are obtained for $x = 0.1$. When x increases to 1, the average altruism ε drops to its minimal value -0.5 almost linearly with x.

For the option C, calculations are made for the three above given initial conditions and the same statistics. In this option, the altruism of each agent remain unchanged, then for the initial conditions *i*) and *iii*) a permanent bias is present in the system towards defection and cooperation, respectively. However, the system evolution (Eqns. 6 and 7) produces another bias, always towards cooperation. Clearly, there is no homogeneous phase here. The obtained values of mean reputations for each of 10^3 systems are practically the same. These results are shown in Fig. 3, as dependent on the parameter z. To demonstrate the character of the latter bias, we show also in Fig. 4 an example of the plots of probability of a common cooperation (R), of a common defection (U), of cooperating but being defected (S) and of defecting a cooperating co-player (T). Plots of the same character were shown in [7] for the symmetric case where altruism is constant. As we see in the plots, in the option C the cooperation is promoted.

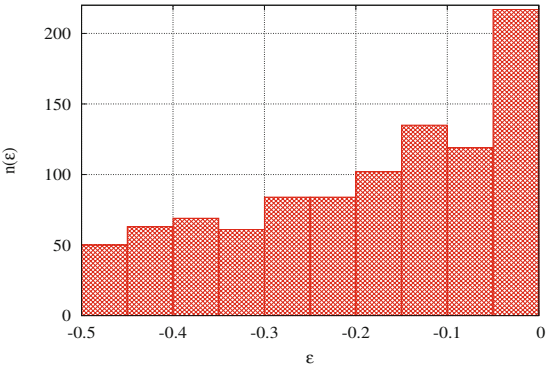


Fig. 2. The altruism distribution $n(\varepsilon)$ for an exemplary system of $N = 10^3$ agents, evolving according to the option B. After 10^5 games, the distribution is almost stable. Here, the initial distribution of altruism promotes defection (case *i)*).

4 Discussion

The result of our simulation is that once the altruism is allowed to evolve, in long time limit the simulated players adopt one strategy, the same for the whole population. This strategy is either to cooperate, or to defect. For the adopted initial distributions of ε_i and W_i, basically the final outcome is determined by the initial mean values $\bar{W} \equiv (V_1 + V_2)/2$ and $\bar{\varepsilon} \equiv (E_1 + E_2)/2$ as follows: once $\bar{W} + \bar{\varepsilon} > 1/2$, the final strategy is to cooperate, otherwise the strategy is to defect. This is true except the case when $\bar{W} + \bar{\varepsilon} \approx 1/2$. In this case it is possible that the whole population defects or cooperates; the respective probabilities vary with $\bar{W} + \bar{\varepsilon}$. This result is new and completely different from the case $x = 0$, considered previously [7]. It is different also from the results obtained above for the model of indirect reciprocity, where altruism does not vary. In the latter model the mechanism which stabilizes cooperation is that agents with small (bad) reputation, even if defected, do not influence the reputation of co-players.

As remarked above, the time evolution of human general attitudes to cooperate is expected to vary slower than the opinions on particular co-players, and it seems reasonable to believe that the former is driven by the latter. We would like to stress that as a rule, what is observed in social phenomena is an interplay of transient effect with different characteristic times. Then, conclusions of modeling should be related rather to the direction of the process than to the stationary state in the long time limit. In particular, our model takes into account a coupling between an agent's experience on the behaviour of the others and the overall willingness of this agent to cooperate. Our results indicate that the feedback is positive; more cooperation bears more altruism what in turn leads to more cooperation. As a rule, an agent's experience that cooperation is met in most cases leads to a general belief that to cooperate is an accepted

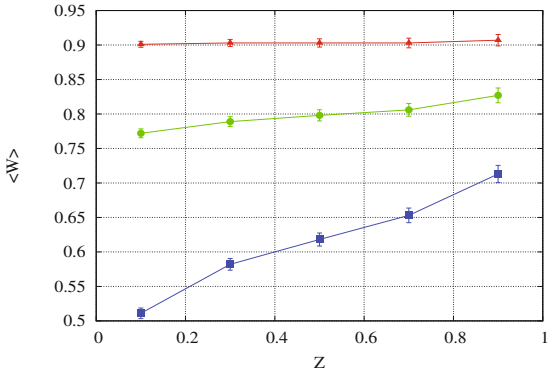


Fig. 3. Mean value of reputation after 10^5 games for a system of $N = 10^3$ agents evolving according to the option C, against the parameter z which controls the speed of evolution of reputation. Three curves are for three cases of initial conditions, *i*), *ii*) and *iii)* from the bottom to the top. The statistics is collected for 10^3 systems.

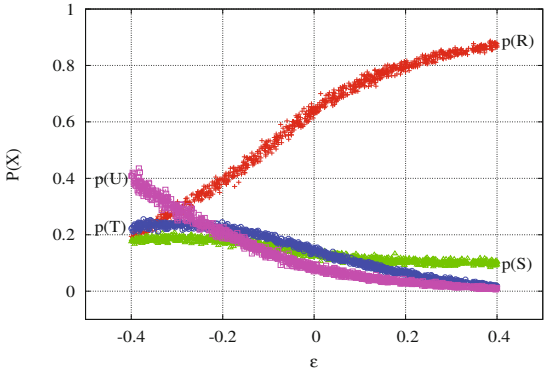


Fig. 4. The probabilities of four outcomes of Prisoner's Dilemma game for the system evolving according to the option C. Here, the initial distribution of altruism is symmetric (case *ii)*). The statistics is collected for an exemplary system of $N = 10^3$ agents after 10^6 games.

social norm. Then, in our theories on experimental data we should consider rather the direction of the process than its stationary stage. One of experiments of this kind was conducted in the Swiss army [13], within the Prisoner's Dilemma scheme. There, platoons of males were formed in a random way for 4-week period of officer training. Having finished the training, individuals believed that members of their own platoons were more willing to cooperate, than others. More data on social experiments can be found in [14].

Acknowledgements. One of the authors (K.K.) is grateful to Hisashi Ohtsuki for a heplful discussion. The research is partially supported within the FP7 project SOCIONICAL, No. 231288.

References

1. Ullmann-Margalit, E.: The Emergence of Norms. Oxford University Press, Oxford (1977)
2. Bicchieri, C.: The Dynamics of Norms. In: Bicchieri, C., Jeffrey, R., Skyrms, B. (eds.), Cambridge University Press, Cambridge (1997)
3. Güth, W., Kliemt, H.: Rationality and Society 10, 377 (1998)
4. Ostrom, E.: J. Economic Perspectives 14, 137 (2000)
5. Roberts, G.: Proc. R. Soc. London B 265, 427 (1998)
6. Van Vugt, M., Roberts, G., Hardy, C.: In: Dunbar, R., Barrett, L. (eds.) Oxford Handbook of Evolutionary Psychology. Oxford University Press, Oxford (2007)
7. Kułakowski, K., Gawroński, P.: Physica A 388, 3581 (2009)
8. Scheff, T.J.: In: Kemper, T.D. (ed.) Research Agendas in the Sociology of Emotions. State University of New York Press, Albany (1990)
9. Turner, J.H., Stets, J.E.: The Sociology of Emotions. Cambridge University Press, Cambridge (2005)
10. Terry, D.J., Hogg, M.A. (eds.): Attitudes, Behavior, and Social Context: The Role of Norms and Group Membership. Lawrence Erlbaum Associates, Mahwah (2000)
11. Bicchieri, C.: The Grammar of Society. The Nature and Dynamics of Social Norms. Cambridge University Press, Cambridge (2006)
12. Ohtsuki, H., Iwasa, Y.: J. Theor. Biol. 239, 435 (2006)
13. Goette, L., Huffman, D., Meier, S.: The impact of group membership on cooperation and norm enforcement, FRB of Boston Working Paper No 06-7 (March 2006)
14. Camerer, C.F.: Behavioral Game Theory. Experiments in Strategic Interaction. Russell Sage Foundation, NY (2003)

Towards Multi-Agent Simulation of the Dynamic Vehicle Routing Problem in MATSim

Michał Maciejewski[1] and Kai Nagel[2]

[1] Institute of Machines and Motor Vehicles, Faculty of Machines and Transportation, Poznan University of Technology, ul. Piotrowo 3, 60-965 Poznan, Poland
michal.maciejewski@put.poznan.pl

[2] Transport Systems Planning (VSP), Institute for Land and Sea Transport Systems, TU Berlin, Salzufer 17-19 Sekr. SG12, D-10587 Berlin, Germany
nagel@vsp.tu-berlin.de

Abstract. The paper presents the idea and the initial outcomes of integrating MATSim, a multi-agent transport simulation system, with the DVRP Optimizer, an application for solving the Dynamic Vehicle Routing Problem. First, the justification for the research is given and the state of the art is outlined. Then, MATSim is presented with a short description of the recent results in areas related to the one proposed in the paper, followed up by the discussion on the DVRP Optimizer functionality, architecture and implemented algorithms. Next, the process of integrating MATSim and the DVRP Optimizer is presented, with the distinction of two phases, the *off-line* and *on-line* optimization. Then, a description of the *off-line* optimization is given along with the results obtained for courier and taxi services in urban areas. The paper ends with conclusions and future plans.

Keywords: dynamic, on-line, vehicle routing, optimization, DVRP, multi-agent, traffic flow, simulation, MATSim.

1 Introduction

The Vehicle Routing Problem (VRP) [1,2] is among the most complex and fundamental components of modern fleet management. From the very beginning, research on VRP was focused on providing solutions aimed at cost efficiency while skipping other aspects of routing. However, nowadays companies must be not only cost effective, but also more open for the customer. Consequently, they offer more sophisticated and flexible services concerning also the timeliness and responsiveness to changing customers' needs. This leads ultimately to the Dynamic Vehicle Routing Problem (DVRP) [3], where the time dimension is one of the main concerns.

Despite the huge technical potential for running dynamic optimization, the existing routing software concerns mainly static routes, and transportation management is mainly performed by human planners and dispatchers, even in large companies [3]. The main problem is that throughout the decades of research on

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 551–560, 2012.

VRP, the great emphasis was put on heuristics accuracy and speed, whereas simplicity and flexibility were out of focus [4]. As a consequence, the best state-of-the-art algorithms give very good results for many theoretical test instances of the static VRP, but they are hard to adapt to the dynamic real-world problems.

Therefore, it is necessary to focus the future research on realistic VRPs. But this requires the development of a system that will be able to simulate various real-world vehicle routing problems thus allowing for both testing optimization algorithms and planning transport services. Such a realistic simulation has to incorporate realistically modelled dynamism of customer demand, traffic flow phenomena and fleet management operations. Especially the optimization of transport services in urban areas is extremely challenging due to high dynamics of traffic flow resulting in continuously changing travel times and often, depending on the type of services, in high volatility of demand (e.g. taxi). Moreover, when considering city-logistics policies, many additional and often mutually conflicting objectives appear as, for example, the reduction of the negative influence on the environment and on the local society, or the positive influence on city development.

In the recent years several approaches that combine vehicle routing and traffic simulation have been proposed and implemented. In one of the first works in this field Regan et al. [5] have proposed a simplified simulation framework for evaluation of dynamic fleet management systems for the truckload carrier operations. Taniguchi et al. [6] have analysed integration of vehicle routing optimization and traffic simulation for optimization of city-logistics oriented problems. Another attempt is an application of the AIMSUN simulator for optimization of the VRP in cities by Barcelo et al. [7]. Liao et al. [8] have developed a system for evaluation DVRP strategies using real-time information. Unfortunately, all these approaches are only partial solutions, as they do not include realistic demand modelling and simulation. This limits their application mainly to problems with static and known a priori demand. As a consequence, it is impossible, for instance, to model the impact of traffic or transport service availability on customer demand. Moreover, the systems were used only for small-scale problems, where a road network was of limited size and the number of customers was small.

2 Multi-Agent Simulation and the DVRP

One possible solution to overcome the deficiencies of the existing solutions is to use a system that allows for detailed modelling of complex interdependencies between the three main components, that is customer demand, traffic flow and vehicle fleet, and that is able to run large-scale simulation. The first requirement implies use of a multi-agent simulation approach that includes all the actors and components, that is *inhabitants* that generate *traffic* in a network and, as a *customers*, create *demand* for services which are provided by *companies* which in turn have *vehicles* that are dispatched to serve the customers' requests and thus also participate in *traffic* generation. The services may be connected with transport of passengers (e.g. taxi) or goods (e.g. courier).

Since fast and realistic traffic flow simulation is the key issue, before choosing a concrete simulation system, four popular traffic simulators, namely MATSim, TRANSIMS, VISSIM and SUMO, had been tested [9,10]. All the systems except for SUMO gave, after calibration, correct results. However, comparing the systems' functionality, MATSim (Multi-Agent Transport Simulation) [11] offers the most comprehensive set of features concerning the research goals. First of all, it is a multi-agent activity-based microsimulation system for daily transport demand analysis. Secondly, due to a fast and efficient traffic simulation, it is able to conduct analyses for large scenarios, even concerning a whole country. Last but not least, MATSim modularity and openness (open-source software) allow for extending and adjusting its functionality to one's needs.

In recent years MATSim has been applied in several research works that are to some extend related with the one proposed. Dobler [12] has looked at the so-called within-day replanning approaches for MATSim. In the resulting implementation, synthetic travellers are able to change their routes and/or their future activity locations. Rieser [13] has investigated and implemented a public transit (pt) module for MATSim, where passengers, beeing able to plan pt routes, can walk to pt stops, where they wait until they are picked up by a corresponding pt vehicle; the pt vehicle drops them off at their destination, from where they continue, either to another pt vehicle or to their next activity. Neumann et al. [14] have implemented a simple control strategy for bus drivers, which implies that a bus would delay itself if the bus behind is delayed. Schroeder et al. [15] have simulated synthetic firms that generate demand for shipments. On the other hand, concerning passenger transport, Ciari et al. [16] have used MATSim for the estimation of car sharing demand. That study does, however, not include an implementation of actual car sharing in MATSim. Overall, despite many advances there is a considerable gap between the current MATSim capabilities and what is needed for the DVRP simulation. This paper will look at this gap in more detail.

3 The DVRP Optimizer

3.1 Properties and Possible Applications

The DVRP Optimizer is a program written in JAVA for solving the DVRP. The optimizer is intended to be as general as possible and to work on customizable instances of the DVRP. The program is constantly being developed and currently supports different versions of the DVRP that can be described as the Dynamic Multi-Depot Vehicle Routing Problem with Time Windows and Time-Dependent Travel Times and Costs. As of now it can be used for the one-to-many (many-to-one) vehicle routing, while the many-to-many problems are only supported if trips are not shared (like in taxi services). Currently, it is only possible to solve problems with hard time windows while soft time windows are not supported.

Concerning the dynamism of requests, the optimizer supports not only on-line submissions but also on-line modifications and cancellations by the customers. A request can also be rejected by the company due to, for instance, a shortage of

resources to serve it. However, operations other than submission of new requests are not included into benchmark instances as this leads to the open problem of results non-comparability (discussed by Flatberg et al. [17]). Thus, it is hard to evaluate the efficiency of the proposed algorithms provided that these operations are permitted.

The next enhancements are connected with the inclusion of time-dependent travel times and costs which are represented as functions of both the departure time from an origin vertex $t_{ij}^D(t)$ and the arrival time at a destination vertex $t_{ij}^A(t)$. Although it is common practice to provide travel time data dependent on the departure time, more elaborated strategies of dispatching vehicles require the provision of travel times depending on the arrival time. By default, the travel time data is provided in a form of 3D matrices, t_{ijk}^D or t_{ijk}^A respectively, where i, j denote a pair of origin-destination vertices while k is a time step within a considered time horizon (the time horizon is divided into time intervals of equal size, e.g. 15 minutes). The travel times are then approximated by linear interpolation from the input data. It is possible to supply only the travel times on the departure time t_{ijk}^D, in that case, the travel times on the arrival time are calculated according to the relation

$$t_{ij}^A(t + t_{ij}^D(t)) = t_{ij}^D(t) \ . \tag{1}$$

In order to obtain correct optimization results it is necessary that the input travel time data meets the FIFO property. It means that if two vehicles start from vertex i and go to vertex j, then the vehicle which starts earlier will always arrive earlier at the destination. Therefore, the input data is validated against the following constraints ($\epsilon > 0$):

$$t_{ij}^D(t + \epsilon) > t_{ij}^D(t) - \epsilon \tag{2}$$

$$t_{ij}^A(t + \epsilon) < t_{ij}^A(t) + \epsilon \tag{3}$$

Another important feature of the DVRP Optimizer is its flexibility in reassigning requests to vehicles. There exist four different pre-defined strategies for reassignment:

- *Never* – a request is never reassigned to another vehicle
- *Only planned routes* – a request may be reassigned to another vehicle only if a vehicle to which the request is already assigned has not started yet
- *Time horizon* – a request may be reassigned unless it is planned to be served within a specified time horizon (e.g. 2 hours)
- *Always* – a request assignment is subject to change as long as the assigned vehicle has not started out for serving it

Selection of the strategy depends on the problem properties. From the drivers' point of view, for example, the most preferable is the first option (*Never*) since they prefer less changeable routes. On the other hand, the more flexible the strategy is (e.g. *Always*) the more cost-effective and customer-oriented routes are. To achieve even higher customizability of the strategies, it is possible to

differentiate requests on whether it is a pickup or a delivery and then to apply a different reassignment strategy to each of these two types of requests.

Due to the considerable flexibility of the model, the current implementation of the DVRP Optimizer can be applied to a wide spectrum of the real-world DVRP examples, such as long-distance courier mail, distribution of heating oil, taxi services, feeder services for the Demand Responsive Transport systems (e.g. mini-buses) and others.

3.2 Algorithms

By default, the DVRP Optimizer uses a memetic algorithm, consisting of an evolutionary algorithm and a local search procedure, for solving the DVRP. Nonetheless, the current version of the system is open for extending it with other optimization approaches, such as tabu search or simulated annealing. The implemented memetic algorithm can be characterized in brief as follows:

- *Genotype coding* – for each vehicle a list of requests to be served
- *Crossing-over* – exchanges routes of a randomly chosen vehicle between two parent individuals; since exchanging routes may lead to invalidity of the both newly created genotypes, a repair procedure is executed removing doubled requests and adding missing ones
- *Mutation* – exchanges requests within/between routes
- *Selection for reproduction* – roulette wheel or tournament selection; both with fitness scaling functions
- *Succession* – with (e.g. elitist succession) or without overlapping populations
- *Termination criterion* – different possible criteria (that may be combined): generations count, convergence threshold, total fitness evaluations count etc.
- *Local search* – steepest ascent hill climbing algorithm with the neighbourhood search operator exchanging requests within/between routes

At the beginning, the memetic algorithm is run to get the initial solution for all *advance* (i.e. known a priori) requests. Then, each time the optimizer is notified of any change in the whole system (such as request modifications, travel times updates or vehicle breakdowns) only the local search procedure needs to be run. The DVRP Optimizer does not use insertion heuristics since their applicability is limited to those cases when only new *immediate* requests may appear while everything else in the system (e.g. already submitted requests, travel times/costs) is static. One should note that as long as changes in the system are not very dynamic, the local search algorithm is sufficient for updating solutions. However, when the dynamism is high, it may be necessary to re-run the whole memetic algorithm.

3.3 General System Architecture

In order to keep all necessary data up to date and to calculate valid and efficient routes, the DVRP Optimizer has to be coupled in a real-time mode with

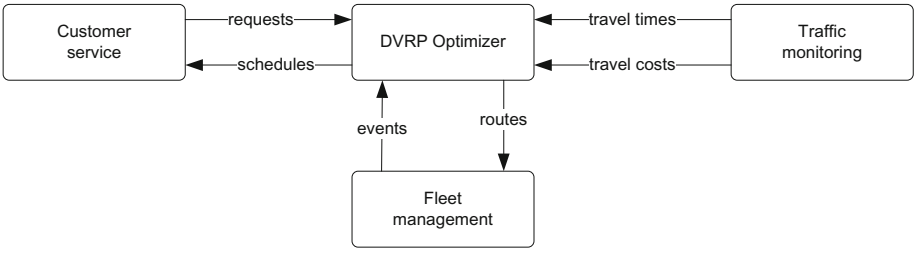


Fig. 1. Data flow between the system components

external modules (Fig. 1). Each of these modules is responsible for a specific domain of the routing process. The *customer service* module informs the optimizer about changes in the customer demand, the *fleet management* module notifies of changes in the state of the vehicle fleet, while the third one, *traffic monitoring*, provides the DVRP Optimizer with the current traffic data and calculates time-dependent shortest paths. Provided that the communication is carried out in real-time, each time any change in the system occurs, the DVRP Optimizer is instantly notified of it, which allows for immediate reaction to a new situation by replanning vehicle routes/schedules. As soon as new plans has been calculated, they are sent to the external modules.

4 MATSim and the DVRP Optimizer Integration

4.1 The Idea

Integrating the DVRP Optimizer with MATSim allows for the simulation-based evaluation of different dynamic vehicle routing algorithms as well as planning of various DRT services in MATSim. In both cases MATSim is responsible for the multi-agent simulation of the transport system with a high level of detail. Besides simulating traffic flow, MATSim serves as a platform for simulating a vehicle fleet and customer demand. Each time a change in demand, traffic conditions or fleet availability occurs, the DVRP Optimizer is informed about it and re-plans routes. After the routes are re-planned, the DVRP Optimizer sends new routes and schedules to MATSim.

The above-presented approach may be implemented in two steps:

1. *Off-line* optimization – combines the time-dependent travel times and costs resulting from simulation in MATSim with external data sources describing supply and demand (i.e. vehicles, depots, customers, requests, etc.); this step requires *sequential* execution of simulation in MATSim and then estimation of the time-dependent travel times/costs prior to the *off-line* optimization
2. *On-line* optimization – runs *concurrently* both the simulation and optimization processes that are integrated and interacting with each other.

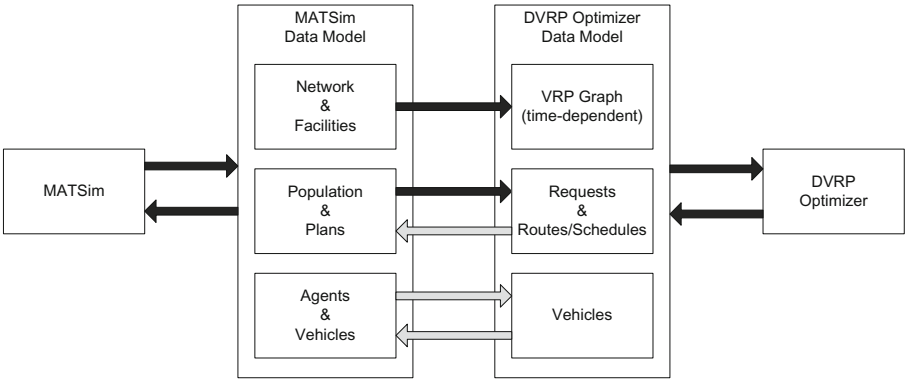


Fig. 2. Data flow between MATSim and the DVRP Optimizer

The data flow between MATSim and the DVRP Optimizer is presented in Fig. 2. The black flows are used in the *off-line* optimization, whereas the grey ones are additionally required for the *on-line* optimization. As of now, the first step, the *off-line* optimization, has been completed and is described later in this section.

4.2 Estimation of Time-Dependent Travel Times and Costs

Prior to the *off-line* optimization, the MATSim simulation results describing the time-dependent link travel times have to be converted into the time-dependent travel times/costs for the VRP graph according to the following steps:

1. Map depots and customers (requests) to links within the MATSim network model. These links form a set of vertices V in the VRP graph $G(V, A)$; a set of arcs A consists of all ordered pairs of the vertices.
2. For each arc (i, j) in A and for each time step k find the shortest-time (or least-cost) path in the MATSim network for a vehicle departing at time step k and calculate the corresponding travel cost c^D_{ijk}. This step is done by means of one of the shortest path search algorithms available in MATSim [18].
3. Based on Eq. 1 calculate travel times on the arrival time t^A_{ijk}. Alternatively, one may use a *backward* shortest path search, which gives more accurate results, but at the cost of longer computation time.

4.3 Results for the Off-Line Optimization

In order to validate the above-presented approach and to assess the efficiency and correctness of the proposed algorithms, a set of tests was designed and carried out (Tab. 1). In the tests, four different networks and two kinds of services were used. Although all the networks were artificially created, their layouts are typical for urban areas, and the generated traffic is also characteristic for cities. Also the requests were designed in such a way that they would imitate real

Table 1. Properties of the test cases

No	Network layout	Nodes	Links	Service type	Requests	Immediate requests	Pickup requests
1	grid	62	170	courier	100	49	49
2	custom	129	326	courier	100	52	52
3	spider's web	57	208	taxi	100	95	n/a
4	custom	93	276	taxi	100	96	n/a

Table 2. Best results obtained for the test cases

No	Type	TD TT	Averaged TT	
1	static	13728	17192	(+25,2%)
	dynamic	14766	17342	(+17,4%)
2	static	22435	26351	(+17,5%)
	dynamic	23879	28133	(+17,8%)
3	static	54017	53778	(-0,4%)
	dynamic	54307	54327	(+0,0%)
4	static	95579	96406	(+0,9%)
	dynamic	96451	97158	(+0,7%)

services within urban areas (e.g. the advance-to-immediate requests ratio, time windows, pickup-to-delivery ratio). The size of a network influences the speed of estimating time-dependent travel times/costs since the computation time of the shortest path search depends on the number of nodes and links; however, in case of the *off-line* optimization, it does not affect the the routing algorithm speed which is dependent (mostly) on the number of submitted requests.

The test procedure was carried out according to the following rules:

- Prior to the *off-line* optimization, for each test case the MATSim simulation was carried out, and then the time-dependent travel times and costs matrices were calculated with a 15-minute resolution. It was assumed that the travel costs are equal to the travel times.
- Tests instances were run both as the *static* and *dynamic* VRP.
- Each test instance was run with time-dependent travel times (*TD TT*) and with constant (a 24-hour average) travel times (*averaged TT*). In the latter case, the obtained results were then re-evaluated using the time-dependent travel times to show the real costs considering changeable road conditions.

Comparing the results for the *courier services* tests (Tab. 2), one can see that the precise knowledge of travel times leads to much better results. By knowing the time-dependent travel times, it is possible to arrange schedules in a way which minimises travelling along arcs at the moment they are at their highest-level of congestion. Moreover, it is very likely that solutions calculated with the averaged travel times may not satisfy all the constraints; in the experiments, almost all solutions violated the time window constraints, and in extreme cases, the total lateness (being after a time window is closed) was almost 1.5 hours. Comparing *dynamic* vs. *static* mode, it is clear that, in case of courier services, having known all the requests in advance, one can yield better results.

Different results were obtained for the *taxi services* tests. First of all, the solutions found for the averaged and the time-dependent travel times are similar concerning the cost of travelling. This is due to the properties of taxi services – taxi services work more in a request-response manner with narrow time windows for serving requests, which makes it impossible to flexibly change the order of requests and thus to avoid traversing arcs when they are congested. This is also one of the reasons of smaller differences when comparing *dynamic* vs. *static* mode than it was for the courier services test cases. However, the real difference lies in constraint satisfaction – the solutions obtained with the averaged travel times were always violating some of the time windows. This means that when using averaged travel times, some of the customers were always served too late.

Since the tests were carried out for instances created specifically for traffic simulation in MATSim, it is impossible to directly compare the results of this research with other studies, however, some similarities may be shown. Both Ichoua et al. [19] and Van Woensel et al. [20], using popular VRP test instances, proved that knowledge of accurate travel times results in substantial improvement of solutions. The scale of this improvement depends on many factors, including the level of time-dependency. In the second study cost decrease was 25% on average, which is similar to the outcomes for the courier services (Tab. 2).

5 Conclusions and Future Work

In this paper, an application of MATSim as a multi-agent simulation platform for DVRP optimization is presented. The approach takes into account several factors which are essential for finding optimal routes in the real world, like traffic in urban areas, and which are missing in the classical VRP models. The results show that the lack of accurate travel-time data may lead not only to higher costs, but also to numerous violations of time-window constraints, which in the end results in the deterioration of service quality.

The implemented *off-line* optimization approach is an important step towards the full integration of MATSim with the DVRP Optimizer. The authors work on creating an *on-line* optimization system that will serve for planning taxi and Demand Responsive Transport services in MATSim, according to the idea presented in Section 4.1. Additionally, it will be possible to adapt such a system to serve as a simulation framework for carrying out very detailed evaluations of different dynamic time-dependent vehicle routing algorithms. Both applications seem very important, appealing and timely.

References

1. Toth, P., Vigo, D. (eds.): The Vehicle Routing Problem. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2001)
2. Golden, B., Raghavan, S., Wasil, E. (eds.): The Vehicle Routing Problem: Latest Advances and New Challenges. Springer, NY (2008)
3. Zeimpekis, V., Tarantilis, C.D., Giaglis, G.M., Minis, I. (eds.): Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies. Springer, NY (2007)

4. Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., Semet, F.: A guide to vehicle routing heuristics. Journal of the Operational Research Society 53, 512–522 (2002)
5. Regan, A.C., Mahmassani, H.S., Jaillet, P.: Evaluation of Dynamic Fleet Management Systems: Simulation Framework. Transportation Research Record 1645, 176–184 (1998)
6. Taniguchi, E., Thompson, R.G., Yamada, T., Duin van, R.: City logistics – Network Modelling and Intelligent Transport Systems. Pergamon, Amsterdam (2001)
7. Barcelo, J., Grzybowska, H., Pardo, S.: Vehicle routing and scheduling models, simulation and city logistics. In: Zeimpekis, V., Tarantilis, C.D., Giaglis, G.M., Minis, I. (eds.) Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies, pp. 163–195. Springer, NY (2007)
8. Liao, T.-Y., Hu, T.-Y., Chen, D.-J.: Object-Oriented Evaluation Framework for Dynamic Vehicle Routing Problems Under Real-Time Information. In: 87th TRB Annual Meeting, Washington (2008)
9. Maciejewski, M.: A comparison of microscopic traffic flow simulation systems for an urban area. Transport Problems 5(4), 27–38 (2010)
10. Maciejewski, M.: Parametric Calibration of the Queue-Based Traffic Flow Model in MATSim. In: Janecki, R., Krawiec, S. (eds.) Contemporary Transportation Systems. Selected Theoretical And Practical Problems. Models of Change in Transportation Subsystems, pp. 217–226. Wydawnictwo Politechniki Slaskiej, Gliwice (2011)
11. Balmer, M., Meister, K., Rieser, M., Nagel, K., Axhausen, K.W.: Agent-based simulation of travel demand: Structure and computational performance of MATSim-T. In: 2nd TRB Conference on Innovations in Travel Modeling, Portland (2008)
12. Dobler, C.: Implementations of within day replanning in MATSim. Arbeitsbericht Verkehrs- und Raumplanung, vol. 598, IVT, ETH Zurich (2009)
13. Rieser, M.: Adding transit to an agent-based transportation simulation concepts and implementation. PhD thesis, TU Berlin (also: VSP WP 10-05) (2010)
14. Neumann, A., Nagel, K.: Avoiding bus bunching phenomena from spreading: A dynamic approach using a multi-agent simulation framework.VSP Working Paper 10-08, TU Berlin, Transport Systems Planning and Transport Telematics (2010)
15. Schröder, S., Zilske, M., Liedtke, G., Nagel, K.: Towards a multi-agent logistics and commercial transport model: The transport service provider's view. In: 7th International Conference on City Logistics, CITY LOGISTICS 2011, Mallorca Island, Spain, June 7-9 (2011)
16. Ciari, F., Schüssler, N., Axhausen, K.W.: Estimation of car-sharing demand using an activity-based microsimulation approach. Working Paper, 632, IVT, ETH Zurich (2010)
17. Flatberg, T., Hasle, G., Kloster, O., Nilssen, E.J., Riise, A.: Dynamic and stochastic vehicle routing in practice. In: Zeimpekis, V., Tarantilis, C.D., Giaglis, G.M., Minis, I. (eds.) Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies, pp. 41–63. Springer, NY (2007)
18. Lefebvre, N., Balmer, M., Axhausen, K.W.: Fast shortest path computation in time-dependent traffic networks. Working Paper, 439, IVT, ETH Zurich, Zurich (2007)
19. Ichoua, S., Gendreau, M., Potvin, J.-Y.: Vehicle dispatching with time-dependent travel times. European Journal Of Operational Research 144(2), 379–396 (2003)
20. Van Woensel, T., Kerbache, L., Peremans, H., Vandaele, N.: Vehicle routing with dynamic travel times: A queueing approach. European Journal Of Operational Research 186(3), 990–1007 (2008)

The Application of Cellular Automata to Simulate Drug Release from Heterogeneous Systems

Agnieszka Mietła[1], Iwona Wanat[1], and Jarosław Wąs[2]

[1] AGH University of Science and Technology, Institute of Process Control
[2] AGH University of Science and Technology, Institute of Automatics,
al. Mickiewicza 30, 30-059 Krakow, Poland
{mietla,wanat,jarek}@agh.edu.pl

Abstract. The simulation of drug release from a heterogeneous system is described in the article. Three phenomena were considered in the simulation: solvent flow through the pores of the media, drug dissolution and diffusion to the external environment. Cellular Automata (CA) were used to model the drug carrier and all processes. New diffusion algorithms were proposed. The simulation and laboratory results were compared and it was shown that they have a high level of similarity.

Keywords: Cellular Automata, controlled drug delivery, diffusion, dissolution.

1 Introduction

Controlled drug release has attracted great interest over the last few years. The basis of this concept concerns the precise delivery of highly accurate drug doses at the appropriate time and place in the human body. In the case of bone disease, it may be necessary to fill the gaps with implants containing appropriate active pharmaceutical ingredient.

The development of computers has created a temptation to replace some of the in-vitro or in-vivo experiments with simulation. Numerous papers dealing with the problem of drug release simulation from homogeneous carriers can be found in [1–6]. Commonly used tools are cellular automata (CA) [7] and the Monte Carlo method. In [1] the authors proposed a 3D cellular automaton to simulate drug release from biodegradable microspheres. This simulation includes a process of material degradation and drug diffusion. The algorithm for diffusion is based on Brownian motions [8] and is called random walk. In [2, 3] the authors used Monte Carlo to create a model where calculations were conducted for a 2D-cross-section of the carrier and a 'life expectancy' parameter was introduced to simulate drug or solid material dissolution. Work on the model was continued by

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 561–569, 2012.

Gopferich and Siepmann. There are far fewer papers concerning heterogeneous than homogeneous systems. One of them is [9], where the authors proposed a model of drug release, forming a 3D multi-layer microstructure with microchambers, based on CA. In [10], the authors present their research on the structural optimization of drug release devices in the form of an array with microchambers.

The purpose of this article, described in detail in Section 2, is to use cellular automata, presented in Section 3, to model drug release from a heterogeneous carrier in the form of a cylinder. In this study particular attention is paid to three main physical processes: solvent flow (Section 4), drug dissolution (Section 5) and diffusion (Section 6). In the model described in this article, the material is non-degradable and the drug is released through pores in the diffusion process. The modified random walk method and the new diffusion algorithms with the Margolus neighborhood have been tested. The random walk algorithm has been extended by the possibility of changing the rate of diffusion.

2 Formulation of the Problem

In the case of controlled drug release, one of the most important pieces of information is the profile of the drug released at that time. This helps to determine how quickly active pharmaceutical ingredient should be released and therefore allows the optimal dose to be chosen. The aim of the project was to create a simulation of drug release from a heterogeneous, cylindrical carrier. Inside the molder, there was a separate phase containing the drug. Such a simulation should return a release profile.Drug release is extremely complex, however, three main physical processes may be distinguished and modeled. Cellular Automata (CA) have been used to model three physical phenomena: the inflow of water through the pores to the interior molder, drug dissolution, drug diffusion outside the molder through the pores.

3 Use of Cellular Automata

In order to simulate the drug release, a cellular automaton was used. A cellular automaton has a two-dimensional lattice, which represents a cross-section of the molder. The CA is constructed on a 200 x 200 lattice. In the automaton, each cell is a square. In the model there are five types of cells: PORE, SOLVENT (buffer), MATERIAL, SOLID DRUG, DISSOLVED DRUG.

The inner circle of radius r = 30 represents part of the drug carrier that contains active pharmaceutical ingredient in a solid form. In contrast, the outer circle has a radius of the value r = 90 and contains cells of MATERIAL. Other cells, located outside the circle, are SOLVENT. DISSOLVED DRUG cells that cross the border of the carrier are summed. This sum divided by total number of DRUG cells is called a **drug release amount**.

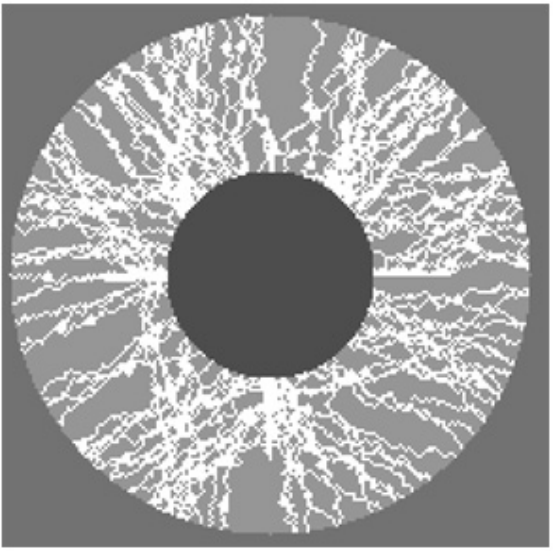


Fig. 1. Cellular automata representing a cross-section of the drug carrier

4 CA Models of Water Flow

There are many ways to describe the fluid motion of the particles [11], for example Euler's method or Lagrange's method. However solving differential equations is always a complex and time-consuming task. Therefore, as a result of attempts to simplify the modeling of fluid flow, a method based on cellular automata is proposed as a substitute for continuous models. With the advent of the lattice-gas model cellular automata [12], there has been a huge breakthrough in the simulation of water flow in the pores. The main method of this group of models is the Lattice-Boltzmann method (LBM).

One of the models for the transport of fluids was developed by Frish, Hasslacher and Pomeaou. It is a two-dimensional model of the lattice-gas known as FHP. The cellular automaton in the macroscopic scale isotropic, represents Navier-Stokes equations for low Mach numbers [13]. The advantage of this model is the simple implementation of complex boundary conditions, which involves the possible application of this method to simulate fluid flow through the porous structure of the medium.

5 Drug Dissolution Algorithm

The aim of the dissolution process is to change the SOLID DRUG cells into DISSOLVED ones. Only then may the diffusion process take place. All solid drug cells in CA have a parameter (a lifetime) corresponding to their solubility, which can take natural values starting from 1. If, at the current iteration, a

solid drug cell has at least one solvent cell neighbor, then the parameter is decremented. When the life time is zero, then the cell state is changed to a dissolved drug.

6 Drug Diffusion Algorithms

Generally, implemented diffusion algorithms can be divided into two groups: on the one hand, methods based on the Moore neighborhood (**Brownian motion, naive diffusion**) and on the other hand, algorithms which operate on the Margolus neighborhood (**HPP-gas, TM-gas, block rotation**). This is an interesting type of neighborhood in which the iteration consists of two parts. Neighborhoods are shown in Figure 2.

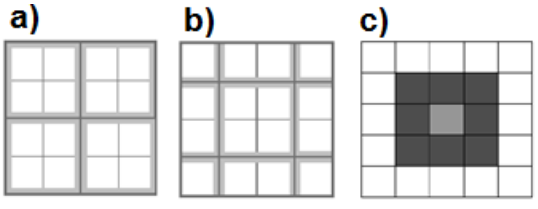


Fig. 2. Margolus neighborhood in even (a) and odd (b) parts of iteration; (c) - Moore neighborhood

6.1 Diffusion on the Moore Neighborhood

The article [1] describes the process of drug diffusion from a spherical, homogeneous carrier. Diffusion is implemented as a random process which imitates Brownian motions. The von Neumann neighborhood is used in the model. During a single iteration, a drug molecule performs 400 position changes with randomly selected SOLVENT neighbors.

To implement a diffusion algorithm in the simulation, the Moore neighborhood was proposed due to its greater number of neighbors (possible directions of particle movements) . In fact, the particle may move in any direction, therefore the choice of the Moore neighborhood seems to be appropriate.

A parameter specifying the **number of particle location changes during one iteration** was added to the program. Changes in this parameter mean a change in the rate of diffusion [14]. In fact, the diffusion rate varies for different physical systems, therefore it is important to take it into account in the simulation. In the simplest case, described in [15], this parameter equals one. This algorithm is called naive diffusion due to [15].

6.2 Diffusion on the Margolus Neighborhood

All the methods are described in [15]. The first, called block rotation, is a non-deterministic method. The transition rule is the same for each part of the iteration: all cell blocks from the current iteration part are rotated by 90 degrees clockwise or counterclockwise (with the same probability). The second method is called TM-gas. If there are two molecules on the first diagonal of a block and two cells in solvent state on the second, there is a collision. If there is no collision, the block is rotated 90 degrees clockwise at even parts and counterclockwise at odd parts [15]. Another method, HPP-gas, is similar to the previous one. Cells from a block change their state (molecules to solvent, solvent to molecules) if there is a collision. In other cases, the block is rotated 180 degrees at both parts of the iteration [15].

Unfortunately, it is impossible to apply these methods to drug release simulation without modification because they would result in MATERIAL and SOLID DRUG cells movement. Models with the Margolus neighborhood need modifications which take interactions between moving drug particles and static obstacles into account. All the combinations that may occur in a cell block were analyzed and new transition rules were added. In the case of the TM-gas algorithm, a new situation has been defined as a collision. If, at current iteration part, drug cell rotation is blocked by an obstacle, then there is a collision. In this situation the rotation is abandoned. Similarly, if obstacles prevent drug molecules rotating 180 degrees in the HPP-gas algorithm, then collision rules are applied. In this situation dissolved drug particles are rotated 90 degrees in randomly chosen directions (if there is a choice). In he Block Rotation algorithm, if an obstacle makes rotation in a drawn direction impossible, then it is continued in the opposite direction. If there are two DISSOLVED DRUG cells in a block and one obstacle, then at least one molecule changes position. New transition rules are simply a description of particle reflection from static obstacles.

7 Results

Before the results are presented, **the drug release velocity** should be defined. This is a change in the quantity of the drug at the time [16]. However, while calculating the first derivative of the data representing the profile of the release, it appears that velocity is not constant. In most cases, it is the highest in the first stage. This part of the profile is almost a straight line. At the end of the release process, velocity gradually decreases. In this work, another definition of velocity has been applied. This is a slope (in degrees) of the profile at a point where there was a 50-percent drug release.

To determine how the number of particle location changes during one iteration in Brownian diffusion affects the release velocity, a series of simulations was carried out. The collected data is shown in Figure 3. Each time the solubility parameter has a value of 1.

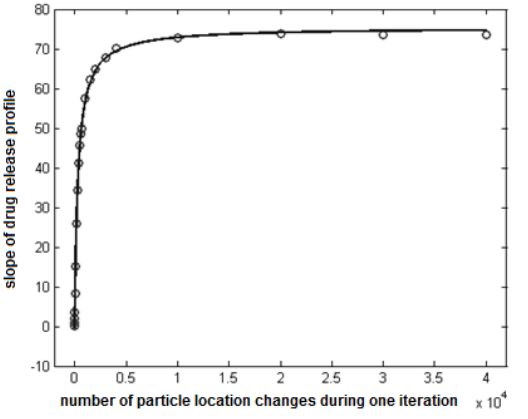


Fig. 3. Relationship between the number of particle location changes during one iteration and drug release velocity in Brownian motion diffusion

It was observed that the following function can be adjusted to the results:

$$s = \frac{-1}{3.89 \cdot 10^{-5} \cdot (x + 337.51)} + 75.37 \tag{1}$$

Where s is the drug release velocity (slope in degrees), and x is the number of dissolved drug particle location changes during one iteration.

The impact on the simulation results of solvent flow through the pores was also examined. The profile of drug release in the system where pores were empty at the beginning of the simulation and the second profile, where pores were already

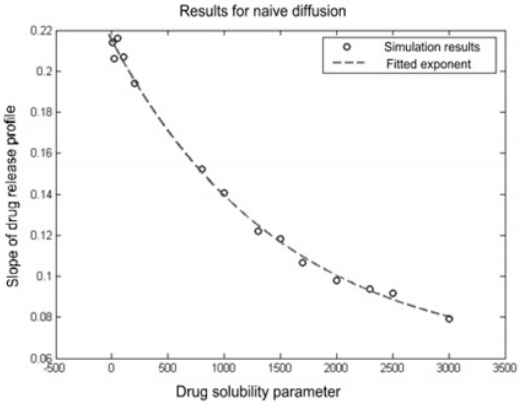


Fig. 4. Relationship between the drug solubility parameter and drug release velocity (naive diffusion method)

flooded, were compared. It appeared that the inflow time is small, compared to the time of diffusion and dissolution, and does not affect the results. In the laboratory experiments the lag time can be observed. This inconsistency between experiments and simulation may be caused by too simplistic model: the pore size and the single solvent particle are to large. This is because of limited CA lattice size. Preservation of original proportion requires much bigger lattice and this affects a computation time.

Drug solubility is the number of milligrams of a substance that can be dissolved in 100 milliliters of solvent. By contrast, the solubility, as a parameter of the program, is the number of iterations during which the DRUG cell must have at least one SOLVENT neighbor to become a DISSOLVED DRUG.

For each diffusion model (Brownian, naive, block rotation, modified TM-gas and HPP-gas) series of simulation for different values of the solubility parameter was carried out. Each simulation was repeated several times to obtain reliable results. For each diffusion method the exponent relationship between the solubility parameter and velocity was obtained. The results for naive diffusion are shown in Figure 4.

It was observed that modified TM-gas and HPP-gas diffusions have a serious drawback - a tendency to block part of the drug inside the carrier. The drug may loop back in pores as shown in Figure 5. The total amount of the released drug is about 90%.

Block rotation diffusion was found to be similiar to Brownian diffusion with low number of particle chages during one iteration.

Simulation results were compared with laboratory tests (drug release from a hydroxyapatite, cylindrical, heterogeneous carrier). Profiles of a drug called pentoxifylline and the simulation results (time-calibrated) are shown in Figure 6. The similarity of profiles was calculated using the discrepancy index [1]:

$$I_c = \left(\frac{\sum_{t=1}^{N} (E_t - S_t)^2 / N}{\sum_{t=1}^{N} (E_t)^2 / N} \right)^{\frac{1}{2}} \tag{2}$$

Where E_t is the laboratory result at the time t, S_t is a simulation result at the time t, N is the number of measurements. I_c is a real value between 0 and 1. Lower numbers (less than 0.2) signify better compatibility of results [1].

The parameter Ic for the data given in Figure 6 was 0.0577.

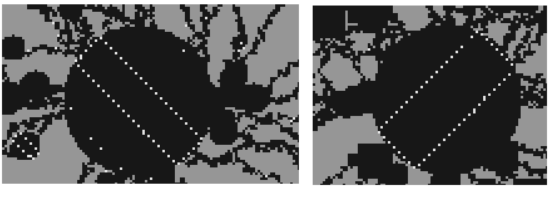


Fig. 5. Drug particle looped back in pores (HPP-gas method)

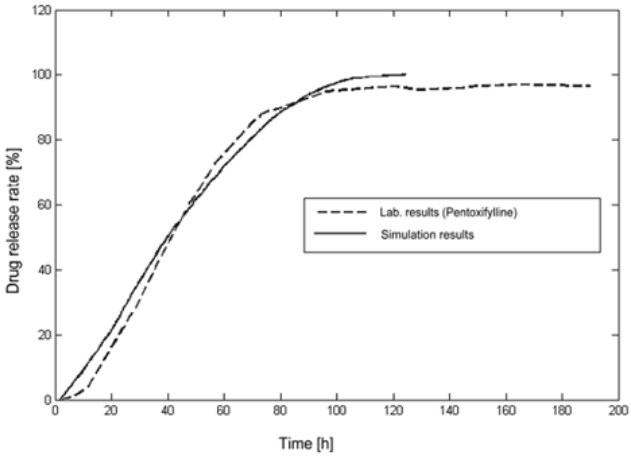


Fig. 6. Comparison of the laboratory results of pentoxifylline released from heterogeneous, cylindrical molder and the time-calibrated simulation results

8 Conclusions

The created program allows drug release from a heterogeneous molder to be simulated. Three main processes necessary for drug release were included: inflow of water through the pores, drug dissolution and diffusion. The program can control diffusion speed (random walk method), solubility and the porosity of the carrier.

Diffusion algorithms on the Margolus neighborhood were developed so that they can be used in systems where static obstacles are present. It was shown that the use of modified HPP-gas and TM-gas methods does not guarantee 100% drug release from the carrier. Block rotation diffusion was found to be similiar to Brownian diffusion but it is harder to implement because of Margolus neighborhood. Therefore the Brownian motion diffusion seems to be a better solution.

The process of solvent flow does not affect the simulation results. Most likely this is caused by too simplified model and to small CA lattice.

The simulation and laboratory results were compared and it was shown that they have a high degree of similarity.

Acknowledgment. Laboratory results were received from Aneta Zima of the University of Science and Technology, Faculty of Materials Science and Ceramics in Krakow, and Beata Mycek of Jagiellonian University, the Faculty of Pharmacy in Krakow.

References

1. Hildgen, P., Bertrand, N., Leclair, G.: Modeling drug release from bioerodible microspheres using a cellular automaton. International Journal of Pharmaceutics 343, 196–207 (2007)

2. Zygourakis, K.: Discrete simulations and bioerodible controlled release systems. Polym. Prep. ACS 30, 456–457 (1989)
3. Markenscoff, P.A., Zygourakis, K.: Computer-aided design of bioerodible devices with optimal release characteristics: a automata approach. Biomaterials 17, 125–135 (1996)
4. Göpferich, A., Siepmann, J.: Mathematical modeling of bioerodible, polymeric drug delivery systems. Advanced Drug Delivery Reviews 48, 229–247 (2001)
5. Heather, J., Barat, A., Ruskin, M.C.: Probabilistic methods for drug dissolution. part 2. modeling a soluble binary drug delivery system dissolving in vitro. Simulation Modeling Practice and Theory 14, 857–873 (2006)
6. Benoit, J., Faisant, N., Siepmann, J.: A new mathematical model quantifying drug release from bioerodible microparticles using monte carlo simulations. Pharmaceutical Research 19, 1885–1893 (2002)
7. Topa, P.: Network Systems Modelled by Complex Cellular Automata Paradigm. InTech (2011)
8. Mazo, R.M.: Brownian motion: fluctuations, dynamics and applications. Oxford University Press (2002)
9. Ruixia, Y., Chen, H., Zhou, X.: Modeling the drug release from 3d multi-layer microstructure with micro-chambers. In: 7th IEEE Conference on Nanotechnology, IEEE-NANO 2007, pp. 558–561 (2007)
10. Jun-min, C., Rui-xia, Y.: Study on structural optimalization of biodegradable polymer controlled drug release microstructure based on ca. In: IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS 2009, pp. 321–324 (2009)
11. Iribarne, J.V.: Atmospheric physics. Panstwowe Wydawawnictwo Naukowe (1988)
12. Rothman, B., Zaleski, S.: Lattice-gas cellular automata. Simple models of complex hydrodynamics. Cambridge Univ. Press (1997)
13. Hasslacher, B., Frisch, U., Pomeau, Y.: Lattice-gas automata for the navier-stokes equation. Physical Review Letters 56, 1505–1508 (1986)
14. Crank, J.: The mathematics of diffusion. Oxford University Press (1979)
15. Bandman, O.B.: Parallel Computing Technologies. Russian Academy of Science, vol. 1662 (1999)
16. Grassi, M., Colombo, I., Grassi, G., Lapasin, R.: Understanding Drug Release and Absorption Mechanisms: a physical and mathematical approach. CRC Press (2006)

Model of Skyscraper Evacuation with the Use of Space Symmetry and Fluid Dynamic Approximation

Wiesława Sikora, Janusz Malinowski, and Arkadiusz Kupczak

Faculty of Physics and Applied Computer Science,
AGH - University of Science and Technology,
Al. Mickiewicza 30, 30-059 Krakow, Poland
Wieslawa.Sikora@fis.agh.edu.pl

Abstract. The simulation of evacuation of pedestrians from skyscraper is a situation where the symmetry analysis method and equations of fluid dynamics finds to be very useful. When applied, they strongly reduce the number of free parameters used in simulations and in such a way speed up the calculations and make them easier to manage by the programmer and what is even more important, they can give a fresh insight into a problem of evacuation and help with incorporation of "Ambient Intelligent Devices" into future real buildings. We have analyzed various, simplified, cases of evacuation from skyscraper by employing improved "Social Force Model". For each of them we obtained the average force acting on the pedestrian as a function of the evacuation time. The results clearly show that both methods mentioned above, can be successfully implemented in the simulation process and return with satisfactory conclusions.

Keywords: symmetry analysis, pedestrian evacuation.

1 Introduction

Analytical models of socio-technical systems which use sets of local parameters may be significantly simplified, when the number of parameters is reduced to smallest number of relevant ones. The symmetry analysis method (SAM) offers such a possibility (described in [1]) because the behavior of social systems under the action of some external conditions may be regarded as similar to the behavior of solid states under the action of temperature or external electric or magnetic fields. Both systems are complex, containing many interacting elements, and are realized in strictly defined spaces. Very often these spaces are strongly restricted, and because of these restrictions not all types of evolutions of these systems are allowed. When these spaces are symmetrical (crystals are good examples of such situation) the symmetry considerations conducting in the frame of theory of groups and their representations are able to predict the types of behavior of the systems, which are permitted by the symmetry of these spaces. The SAM [1] was successfully used for many years to significantly simplify the descriptions of different type of phase transitions in crystals. The discussion of coexistence of

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 570–577, 2012.

different types of system behavior, leading to different properties of the system are based on the assumption, that different functions describing these properties of the system should have the same symmetry. From the theory of representation we know, that these functions should belong to the same irreducible representation of space symmetry group. It gives the powerful instrument to discussions of complex behavior of the systems, because we have the possibility to extract all limitations of free parameters needed for description of the models. The local parameters such as the resultant force acting on each pedestrian and their velocities are related by the symmetry conditions and expressed by the smallest number of relevant coefficients.

The SAM was applied to investigate what is the influence of space symmetry on the quality of crowd evacuation plans and number of model parameters used in crowd behavior simulation for a football stadium and one floor of multi-floor skyscraper [2],[3] and football stadium [4]. The simulations were performed within the Social Force Model designed by Helbing in 2000 [5], developed by J. Malinowski. He introduced in the program the symmetry adapted vector field calculated by the symmetry analysis method as one of the forces acting on the pedestrian during evacuation and the possibility of evacuation from complicated space [3]. The last, new version of the program offers possibility of introducing individual, physical parameters of the agents. (Appendix).

In this paper the problem of evacuation from multi-floor skyscraper is discussed. As the important parameter for safe evacuation not only the time of evacuation, but also the average force acting on individual during evacuation is regarded. Such discussion is possible only by using the developed Helbing method, which for big number of individuals distributed on many floors of skyscraper is time-consuming. Taken into account the translational symmetry of skyscraper the discussion of evacuation may be limited only to one "cell" of the structure - it means only to two floors connected by staircases (different geometry depending on the building architecture is possible) - independently on the number of skyscraper floors. This allow to perform the calculations with smaller number of individuals which significantly reduce the amount of parameters used in simulations and in this way the time of simulation and to apply the results to the whole building.

2 The Analysis of Skyscraper Evacuation

The translational symmetry used in simulation leads to significant reduction of its time. There is no need to consider the whole building during calculations, but only "one unit cell" of it and what we mean by unit cell is shown in the Fig.1 below it is simply two identical floors that are next to each other and connected by the staircase.

As it can be seen changes in the number of unit cells used in simulations do not have an influence on the average force acting on the pedestrian so the translational symmetry has done its work - there is a clear, visible advantage of using it in the calculations.

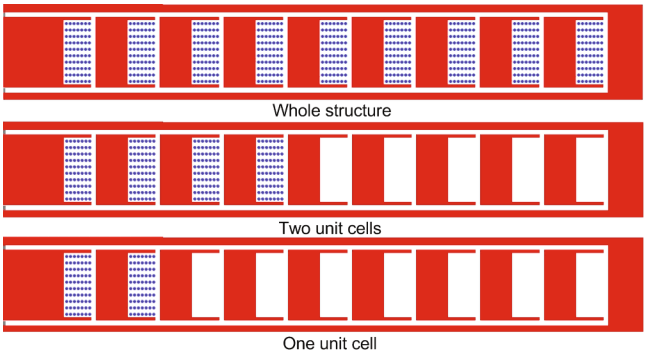


Fig. 1. "Unit cells" of the skyscraper

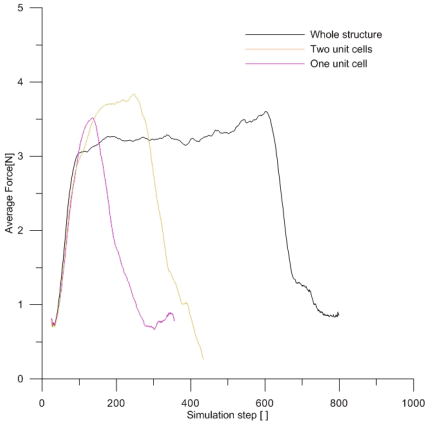


Fig. 2. Average force per pedestrian as a function of evacuation time for different number of unit cells

The constant density of agents which ensures comfortable evacuation of skyscraper may be obtained by introducing time shift between start of evacuation from different floors. The optimal value of this time shift may be calculated from continuity equation as the function of building geometry parameters and social system parameters (for example the number of pedestrian on each floor, their possible velocities...). In this simple model because of collective movements of agents, similarly to the description of charge movement in a crystal under electric field drift, the average quantity of vector velocity (speed) of each agent at the staircase vs is assumed to be constant and the same for each agent. The directions of agents vector velocities stay in agreement with, recalculated by improved Helbing model, fields of displacements.

The time shift between the beginning of evacuation of first floor and the next floor, which guarantees the smooth evacuation with constant density of evacuated individuals, should be:

$$\Delta t = t_f - t_s \tag{1}$$

where t_f is the time of evacuation of one floor (discussed for example in [3]), t_s is the time needed for leaving the staircase which connect two floors. The evaluation of t_f for a given number of pedestrians located on a specific floor with symmetry considerations, was discussed in [3].

The continuity equation and geometry of the building leads to the relations:

$$v_f * c_f = v_s * c_s; v_s = v_f * \frac{c_f}{c_s} \tag{2}$$

where v_f and v_s are speeds of agents at the exit from the floor and at the staircase correspondingly, c_f is the width of exit and c_s is the width of the staircase.

The time t_s needed for covering the staircase length l_s is given as:

$$v_s * t_s = l_s; t_s = \frac{l_s}{v_s}; l_s = \frac{l_s * c_s}{v_f * c_f} \tag{3}$$

then

$$\Delta t = t_f - t_s = t_f - \frac{l_s * c_s}{v_f * c_f} \tag{4}$$

Evaluation of this time shift and sending the information about the accident to given floor at proper time requires high quality ambient intelligence device (AID), which is able to follow and keep in memory the number of pedestrian on given skyscraper floor, and to use this information for determination of floor evacuation, and appropriate time shift.

In this work the simulations of evacuation with calculated time shift are realized in skyscrapers with different staircases. In each case the average force acting on the agent as the function of evacuation time is calculated and compared with similar results got from simulations when the evacuation starts from different floors at the same time. The simplest models, with the same width of exits and staircases are investigated here. The results are presented at the figures showed below.

Fig.1 and Fig.2 show two types of evacuation routes (the difference lies in the length of escape path) which can often be found at the back of the building - fire escape ladder. On the other hand in the Fig.3 we deal with a standard staircase that can be seen in the block of flats as well as in dormitory. The helical staircase in Fig.4 differs from the one in Fig.3 only in the shape of the stairs.

3 Summary

As it can be seen in the presented figures imposing the time shift between start of the evacuation from different floors makes the time of the evacuation longer,

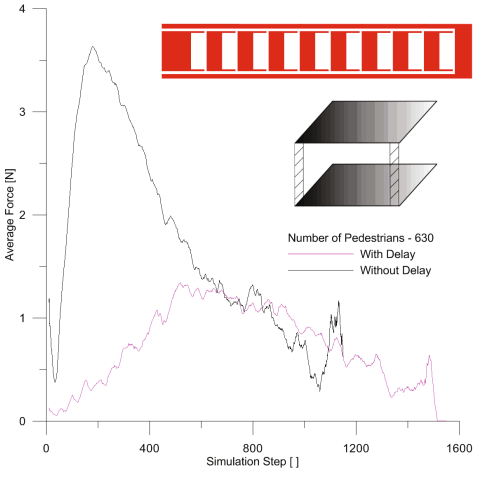


Fig. 3. Average force per pedestrian as a function of evacuation time - fire escape ladder

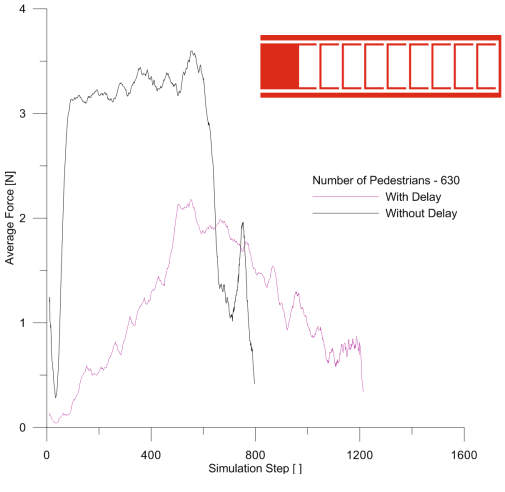


Fig. 4. Average force per pedestrian as a function of evacuation time - fire escape ladder

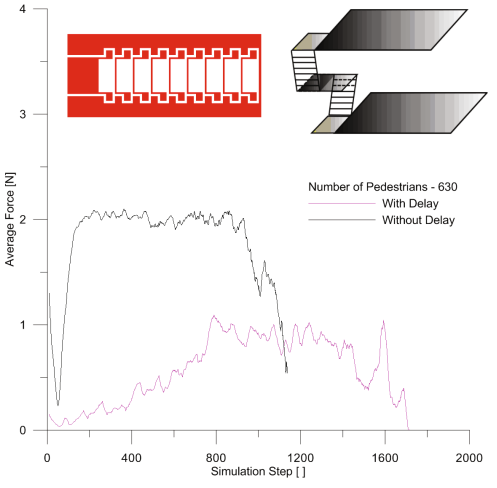


Fig. 5. Average force per pedestrian as a function of evacuation time - standard staircase

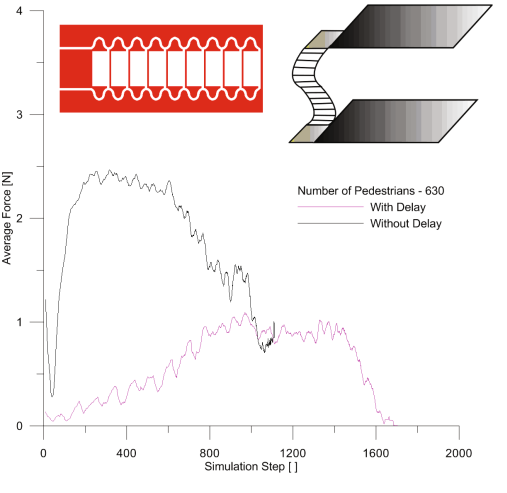


Fig. 6. Average force per pedestrian as a function of evacuation time - helical staircase

but significantly reduces the average force acting on the pedestrian. The good organized evacuation should optimally balance with these two parameters. The reduction of press acting on evacuated induced by properly predicted delay in the information about start of evacuation may be realized for different type of staircases. Such prediction occurs possible by using AMI devices and simplification of calculations following from taken into account the symmetry of the skyscraper. Therefore there is a visible need to put more effort in such investigations, so that the intelligent ambient devices found their places in future building industry.

Simulations with different evacuation parameters will be realized in the future.

Appendix. The evacuation of the skyscraper is simulated by using the Helbing "social forces" model [5], which was taken as a starting point to model presented in this paper.

$$m_i \frac{d\boldsymbol{v_i}}{dt} = m_i \frac{v_i^0 \boldsymbol{e_i^0}(t) - \boldsymbol{v_i}(t)}{\tau} + \sum_{j(\neq i)} \boldsymbol{f_{ij}} + \sum_{W} \boldsymbol{f}_{iW} \tag{5}$$

Equation 5 on right side has a sum of three forces acting on human during evacuation simulation. Second and third one describe sum of repel forces from all other mans and sum of repel forces from all walls and all obstacles inside the evacuated space. First force is the most interesting at this stage. In general it describes in which direction each person should go. In simple case like empty room with some doors, this force can be easily calculated as a sum of attract forces from all doors. Here is also the place for the implementation of vector field calculated by symmetry considerations. This assumption is inadequate with more complicated geometry of the building. Figure 7 shows the example when such approach leads to situation when pedestrian will stick for eternity in dead end.

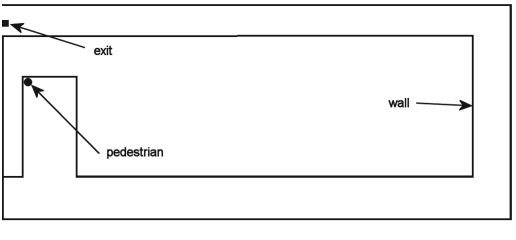


Fig. 7. The "dead-end"

In approach presented in this paper floor space was divided into cells (this got nothing in common with "cellular automata" approach to such kind of simulations!). Every cell can contain obstacle, free space or exit. Every cell that contain free space have vector of desirable velocity connected with it, calculated according to schema described below. Any person that stands inside particular

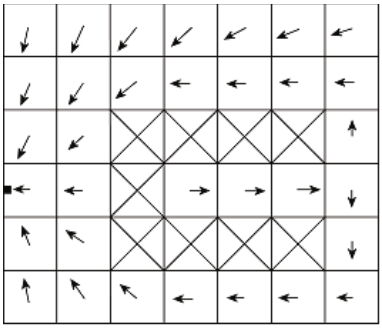


Fig. 8. Final field of velocities

cell takes this vector as his desirable velocity. Unlike in cellular automata approach in this approach "cell" can contain one or more persons and any direction of velocity. Single cell size depends on building geometry only. To make a simulation of an evacuation of single room, small number (like 7x6 for example) of cells is more than enough. Figure 8 show example of such table presenting the final field of velocities. These vectors can be calculated using ï£¡ray castingï£¡ method as shown in paper [6].

Acknowledgement. This work is partially supported by EU program SOCIONICAL (FP7 No 231288).

References

1. Sikora, W., Pytlik, L.: Application of Symmetry Analysis to Description of Ordered Structures in Crystals. In: Group Theory: Classes, Representations and Connections, and Applications, ch. 1, pp. 1–40. Nova Science Publishers, NY (2010)
2. Sikora, W., Malinowski, J.: Symmetry Analysis in Parametrization of Complex Systems. Journal of Physics: Conference Series 213, 012007 (2010), doi: 10.1088/1742-6596/213/1/012007
3. Sikora, W., Malinowski, J.: Symmetry Approach to Evacuation Scenarios. In: Jędrzejowicz, P., Nguyen, N.T., Howlet, R.J., Jain, L.C. (eds.) KES-AMSTA 2010, Part II. LNCS (LNAI), vol. 6071, pp. 229–241. Springer, Heidelberg (2010)
4. Sikora, W., Malinowski, J.: Symmetry in behavior of complex social systems - discussion of models of crowd evacuation organized in agreement with symmetry conditions (submitted) (arXiv:1102.1261)
5. Helbing, D., Molnr, P.: Social force model for pedestrian dynamics. Physical Review E 51, 4282–4286 (1995)
6. Kretz, T., Bonish, C., Vortish, P.: Comparision of Various Methods for the Calculation of the Distance Potential Field, arXiv:0804.3868v1 [physics.comp-ph]

Graph of Cellular Automata as a Metaphor of Fusarium Graminearum Growth Implemented in GPGPU CUDA Computational Environment

Paweł Topa[1,2], Maciej Kuźniar[1], and Witold Dzwinel[1,3]

[1] AGH University of Science and Technology, al. Mickiewicza 30, Kraków, Poland
[2] Institute of Geological Sciences, Polish Academy of Sciences, Biogeosystem Modelling Laboratory, Research Centre in Cracow, Senacka 1, Kraków, Poland
[3] WSEiP High School of Economy and Law, Jagiellonska 109A, Kielce, Poland

Abstract. *Fusarium Graminearum* is responsible for Fusarium head blight (FHB) infection which reduces world-wide cereal crop yield. As a consequence of mycotoxin production in cereal grain, it has also serious negative impact on both human and animal health. The main objective of this study is to develop a mechanistic and conceptual metaphor of Fusarium growth. Our model is based on a new realization of Graph of Cellular Automata paradigm (GCA) used before for simulating anastomosing rivers and the process of angiogenesis in solid tumors. We demonstrate that GCA model is a very universal metaphor, which can also be used for mimicking Fusarium type of growth. To enable 3-D interactive simulations of realistic population ensembles (10^5-10^7 plant and fungal cells), the GCA model was implemented in GPGPU CUDA environment resulting in one order of magnitude speedup.

Keywords: Cellular Automata, GPGPU, complex networks, fungi modelling.

1 Introduction

Fusarium Graminearum is one of the main causal agents of Fusarium head blight (FHB) infection. It attacks cereal crops resulting in a significant grain yield reduction. The epidemic, which took place in North America from 1998 to 2000, costs almost $3 billion. Another effect of this plague is the contamination of grain with mycotoxins, which is extremely harmful for animals and humans.

The infection is initiated by deposition of spores on or inside spike tissue [1]. Wheat heads are the most susceptible to infection during anthesis. Other factors favoring infection are high humidity and temperature. Initially, the fungus does not penetrate the epidermis. As shown in [1], at this stage it develops on the external surfaces of florets and glumes and grows towards susceptible sites within the inflorescence. Other roads of colonization of internal tissue include stomata and underlying parenchyma, partially or fully exposed anthers, openings between the lemma and palea of the spikelet or floret during dehiscence and through the base of the wheat glumes where the apidermis and prenchyma are thinwalled [1].

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 578–587, 2012.

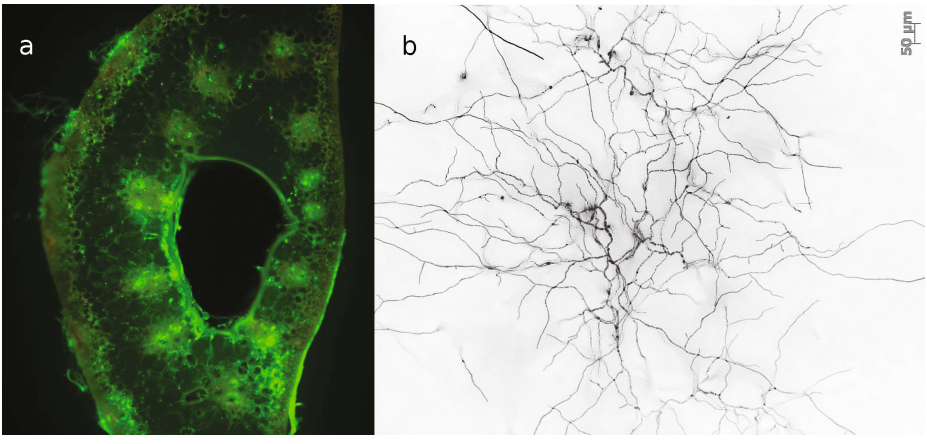


Fig. 1. The images demonstrating Fusarium Graminearum penetrating wheat stem (a) and proliferating on nutrient rich agar well (b). The stem cross-section is displayed. The fluorescent, bright spots correspond to infected sites. (courtesy of Dr Shea Miller and Dr Margaret Balcerzak).

Several mechanisms contribute to the formation of mycelium inside plant tissue. Fusarium growth is highly polarized and hyphae can extend at their tips only. It makes that hyphae is able to attain very high extension rates. Mycelium is able to adapt to local conditions by transporting nutrients from regions where they can be absorbed (living tissue) to the starving sites. A part of mycelium that cannot receive enough nutrients can be degraded and the whole colony can break down onto separated fragments. All these properties make that fungi are very flexible and can actively search the environment for nutrients. To investigate the character of this growth in various conditions and develop new species of grain which are resistant on Fusarium infections, laboratory experiments should be guided by the results from computer simulations. In fact, there are no exact models of biological phenomena due to inevitable simplifications of complex and nonlinear behavior and difficulties in perfect matching of model parameters. However, even approximated computer models allow for fast penetration of parameter space narrowing regions of interest for laboratory experiments.

Cellular Automata is a very efficient modeling paradigm for simulating variety of natural phenomena [2]. The most popular CA system is represented as a regular mesh of cells. The cells change their states according to predefined rules of local interactions. The automaton state depends on the states of its nearest neighbors. Cellular Automata is inherently parallel - all the states of CA system can be updated simultaneously. It gives the opportunity to achieve high computational speed on SIMD (*Single Instruction Multiple Data*) type of architectures e.g. [5], [6] such as GPGPU processors. It is worth to emphasize that Cellular Automata can be treated as kind of framework that can be modified to meet requirements of modelled phenomena (e.g. [3], [4]).

Modern GPUs contain up to hundreds of processing cores. A single GPU can outperform the modern CPUs in solving variety of SIMD implementable problems [7]. GPGPU computing has received great attention after introducing Nvidia CUDA technology [7] and OpenCL standard [8]. Both these environments allow for GPU programming using C-like programming language as well as libraries and drivers that manage programs that run on graphic cards.

Dense mycelia located in a uniform environment, e.g., in Petri dishes, can be represented by continuum density fields of Hyphal distribution and substrate concentration. Thus, the Mycelium proliferation at large spatial scales can be modeled by solving numerically a system of partial differential equations (e.g. [9]). However, when mycelium network is sparse and grows in inhomogeneous environment with non-uniformly distributed substrates, the application of discrete modeling is much more appropriate.

Discrete models based on Cellular Automata were applied for simulating fungal growth several times, e.g., [10], [11]. Boswell at al. [12] proposed a hybrid cellular Automata model. His model uses cellular automata to represent hyphae network but for substrates distribution a continuous approach is applied. All CA models exploit relatively simple domain for mycelium development such as soil. The case, in which the environment has more sophisticated spatial structure, such as wheat stalk or ear of grain, was not considered yet.

In this paper we propose a model based on a Cellular Automata paradigm. We employ the Graph of Cellular Automata [13], [14], [15], which is an extension of classic Cellular Automata approach. The model was tested in simple domains in 2D and 3D spaces. We present the issues of GPU implementation of the model and the speedups obtained.

This paper is structured as follows. Section 2 contains description of the model including explanation of the idea of the Graph of Cellular Automata. The following section contains simulation results with comments. At the end we summarize the conclusions.

2 Model Description

The model of Fusarium that we present here, utilizes the Graph of Cellular Automata modeling tool, which combines Cellular Automata with graphs. This approach was developed for modeling anastomosing river systems [13]. Later it was generalized to cover a class of phenomena that consist of a consuming (or producing) environment and transportation network that supplies (or removes) nutrients to/from the system. The approach was also successfully applied for modeling tumour-induced angiogenesis [14], [15].

While the CA mesh models the environment, the graph of CA represents a transportation network. The graph of CA is constructed over the regular mesh of CA by defining additional relationships between neighboring cells (see Figure 2). All the transportation processes are modeled in the network of connected CA cells – the graph of cells. Diffusion and migration processes occur in the CA environment while linear transportation phenomena in the CA graph. Graph and

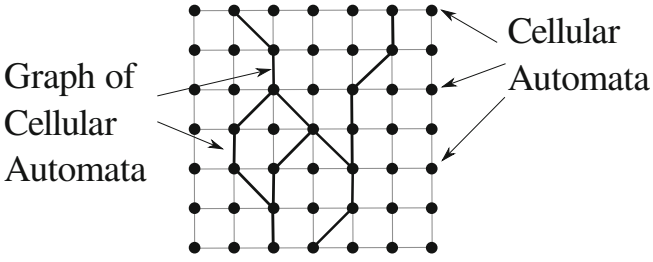


Fig. 2. Graph of Cellular Automata is constructed over regular mesh of CA by establishing additional relationships between the CA mesh and the CA graph

regular mesh are processed separately in a different manner (asynchronously and synchronously). The CA cells, that belong to the CA graph, are the pin points between these two parts of the model.

In this paper we present how the Graph of Cellular Automata can be used to model infection proliferation of Fusarium Graminearum in a plant tissue. The mycelium of Fusarium Graminearum is represented by the graph of cells. The plant tissue is modeled by a regular mesh of Cellular Automata. Mycelium develops its network by invading sites of the plant tissue, what is represented by cells of the CA mesh attached to the CA graph.

Plant tissues covered by the mycelium are poisoned by mycotoxins. Then, the plant cells are killed and all the nutrients are drained out. The mycelium can be degraded locally as the result of lack of nutrients. This part of network will be removed after a short period of time or transmuted into spores. The nutrients can be transported due to diffusion inside mycelium network from the regions rich with nutrients to the starving parts.

We made the following assumptions to the model:

- both Fusarium and plant need nutrients to survive,
- the distribution of nutrients inside a single cell is homogeneous,
- the hyphal tips of the fungus grow randomly, mainly in a straight lines with small directional changes (similarly as in [12])
- mycelia cells are immobile,
- the branching of existing hyphae is a random process (following [16]),
- branching could happen in the network nodes only,
- Fusarium produces enzymes responsible for degrading the cell walls [1], [12] at a constant rate,
- there are two mechanisms responsible for nutrients transportation inside Fusarium: passive (diffusion) and active directed toward the tips,
- nutrients might be absorbed from the plant only when the corresponding cell walls have been already degraded.

The plant cells (environment – represented by CA mesh) are characterized by the following states (at time step t):

- Sp_{ij}^t - the CA state (possible values are: Healthy, Attacked, Destroyed, Dry)
- Np_{ij}^t - the amount of nutrients

- A^t_{ij} - the amount of enzymes degrading the cell walls
- At_{ij} – a threshold over which the cell walls are destroyed
- Pp_{ij} – nutrients production or consumption rate
- Dp_{ij} – nutrients diffusion coefficient in the plant
- Dpe_{ij} – enzymes degrading the cell walls diffusion rate

Fusarium cells (CA graph) are characterized by the following states:

- Sf^t_{ij} – the CA state (possible values are: Tip, Active, Inactive)
- Nf^t_{ij} – the amount of nutrients
- Cfl_{ij} – nutrients consumption rate
- Nft_{ij} – nutrients threshold below which the Fusarium cell becomes inactive
- Dfp_{ij} – passive translocation rate
- Dfa_{ij} – active translocation speed
- $Dfac_{ij}i$ – active translocation cost
- Us_{ij} – nutrients uptake rate
- Uc_{ij} – nutrients uptake cost
- Afp_{ij} – rate of production of the enzymes degrading the plant cell walls
- B_{ij}– probability of branching
- G_{ij} – probability of growing

The algorithm has a simple structure, which is typical for the Cellular Automata approach, (see Fig. 3). In the consecutive steps, the procedures that represent both biological and transportation processes, are executed.

The following processes are included in the model (see Figure 3):

- Fusarium development: A network of mycelium is formed by creating and developing sprouts.The process of growth of each sprout is stimulated by its "tip" motion. The "tip" is located at sprout end. Each "tip" can grow at acute angle to the direction of hyphal growth. The probability of such event depends on nutrients in the hyphae and parameter G_{ij}. Active hyphae cell might branch and form a new sprout with a probability proportional to nutrients and parameter B_{ij}.

  In our model, the "tip" is represented by a cell that belongs to the CA graph with state Sf^t_{ij} marked as **Tip**. The "tip" cell is located at the sprout top and it does not have any successor. The sprout grows by choosing a new cell in the neighborhood of the "tip" cells and the new edge is formed between the old and a new "tip" cell (procedure `graphGrow()`). The **Tip** state label is removed from the previous "tip" cell and given to the new one.

  When nutrient density in Fusarium cell (Nf^t_{ij}) drops below a given threshold Nft_{ij}, the cell alters its state into **Inactive** and after some (randomly chosen) period of time it is removed from the CA graph. This process may result in mycelium network fragmentation.
- Enzymes production:
  Fusarium produces enzymes (mycotoxins) that destroy immunological mechanisms and secretes acid fluid which dissolve the walls of cells. It opens

the way to the nutrients located in the cell body. Active cells that belong to the CA graph produce enzymes and acid fluids at a constant rate $Afp_{ij} * \delta t$ (procedure `graphNodesProduceConsume()`). The enzymes and acid substances are distributed to the neighborhood via diffusion with diffusion coefficient Dpe_{ij} (procedure `cellsDistribute()`). It establishes a gradient of enzyme/acid saturation in the vicinity of the CA graph cells. When enzymes/acid concentration in the plant cell is higher than a given threshold At_{ij}, the cell becomes "Destroyed" and the nutrients uptake is possible.

– Nutrients uptake and transport inside Fusarium: The CA graph cell, that is marked as "Active", uptakes nutrients from the corresponding CA cell only if this cell is in the "Destroyed" state. It means that nutrients can be transferred from the plant cell to the mycelium network. The cost of uptake is non-zero i.e. not all nutrients are passed to the mycelium.

  The nutrients are transported inside mycelium (`graphNodesDistribute()`) via two transport mechanisms: passive — 1D diffusion — and active — the motion of nutritive substances towards the tips. Transport occurs only between those graph cells that are marked as "Active".

The model has been implemented in 2D and 3D spaces. In both cases, the simulation procedures are the same except some implementation issues of the access to the data structures.

At this stage we ignore all defense mechanisms of the plant as well as their interaction with toxic substances that impair these mechanisms. In all the simulations which results are commented in the following section, the plant cells neither produce nor consume nutrients ($Pp_{ij} = 0$)). Diffusion of substances in plant tissue was also omitted in order to simplify the model ($Dp_{ij} = Dpe_{ij} = 0$).

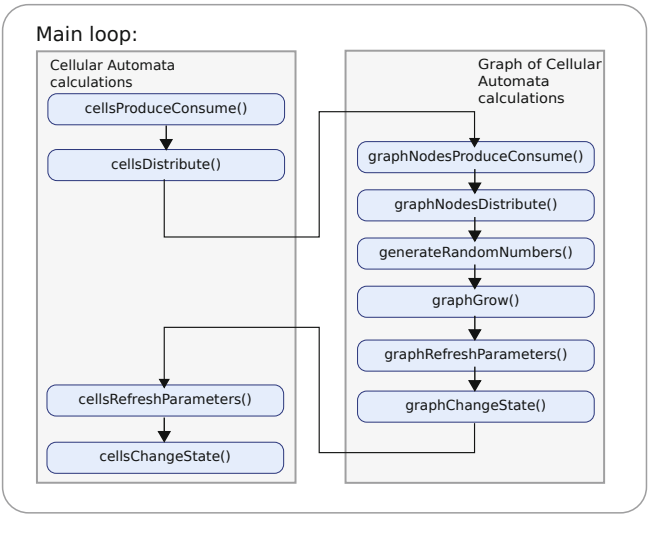


Fig. 3. The block scheme of the main part of the model implementation

2.1 Implementation

Initially, the model was implemented in C++ on a single CPU in a serial mode. This version was designed to test the coherence and compatibility of model assumptions and provides a reference point to its GPU implementation. Simulation results are visualized by using the Amira package [www.amira.com].

The targeted implementation environment is GPU board with CUDA interface. Each procedure presented in Figure 3 is executed as one CUDA kernel. In the prototype implementation all data required by the code are kept in the slow global memory. In case of it extensive use, large delays can be expected. The common approach to increase the efficiency is employing much faster shared memory, which is located on the GPU chip. The limited access to data stored is the cost of such the operation. Data located in the shared memory is accessible by other threads in the same block only. On the other hand, the CA model implementation does not require many data reads and writes. Most variables are read just once, processed and than stored again. In such the case the shared memory does not help too much to achieve greater efficiency. Therefore, to avoid sophisticated and troublesome coding the first approach, exploiting global memory, was implemented.

3 Results of Simulation

Figures 4a and 5 demonstrate simulation results in 2D and 3D spaces and visualized with Amira software. Initially, nutrients are uniformly distributed in the whole domain. The plant cells invaded by mycelium are also attacked with mycotoxins and acid substances. After dissolution of protective layer, nutrients are absorbed by mycelium. In Fig.4a we show the snapshot from simulation in 2D space compared to the microscopic image of Fusarium Graminearum 22 hours after inaculation growing in similar nutrient rich environments (Fig. 4b). One can observe visual similarity of these two pictures. In Fig.4a the mycelium

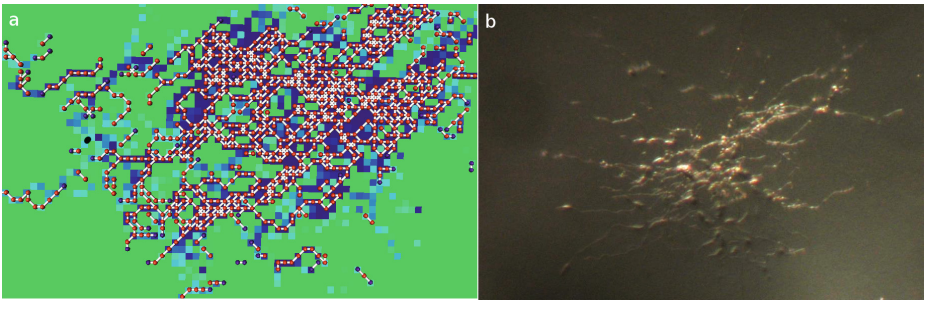


Fig. 4. a) A snapshot of mycelium network generated by the model. Dead areas of plant tissues are marked in blue. b) The microscopic image of Fusarium Graminearum growing in a nutrient rich environment (courtesy of Dr Shea Miller).

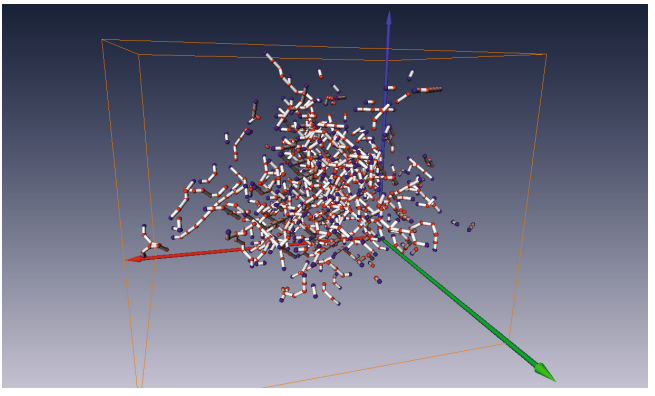


Fig. 5. A snapshot from 3D simulation. Plant tissue is not visualized for clarity.

Table 1. Average execution time and speed-ups obtained for optimized GPU code version versus CPU implementation. Tests were performed for various size of CA mesh. The time was measured from the start to the moment where Fusarium network reached certain size: 5000 nodes, 100000 nodes and an optional size after 5000 steps of simulation.

CA size	**Network nodes**	**Optimized CPU [s]**	**GPU [s]**	**Speedup over optimized CPU**
2500	5500	14.906	0.48	31.05
2500	100000	189.234	0.52	363.91
10k	5400	13.769	0.57	24.16
10k	109000	212.969	0.64	332.76
10k	500000	911.109	0.72	1265.43
40k	5000	16.016	0.95	16.86
40k	103000	205.563	1.04	197.66
40k	625000	1159.047	1.19	973.99
160k	5000	23.36	2.51	9.31
160k	100000	202.188	2.64	76.59
160k	555000	1039.531	2.85	364.75

network is presented together with distribution of nutrients in the plant tissue. The areas of reduced nutrient concentration are marked in blue. The mycelial cells may be degraded if they do not receive sufficient nutrition level. We can see several small fragments of mycelium that surrounds one large network. The large network can survive due to transport of nutrients inside mycelium body. As shown in Fig. 5, in 3D domain and nutrient rich environment, the mycelium network forms a globule of uniform density.

Several performance tests were conducted in order to measure the speedup of the computations. In each case a simulation with different starting parameters

has been carried out. The performance of prototype CUDA application utilizing the NVIDIA GeForce GTX 295 graphics card was compared with an optimized C++ program run on a personal computer with a 2.16 GHz Intel processor (single thread execution). As the execution path is random and computation time highly depends on the number of graph nodes, average results from multiple runs have been collected. The tests were performed for one set of configuration parameters. Simulations were conducted until the Fusarium network reached a certain size (about 5000 nodes and 100000 nodes). Additionally, other tests were run through 5000 time-steps and final size of Fusarium network was considered. Table 1 collects averaged execution times and speedup measured for this test. The results are very promising. Speedups of at least one order of magnitude are observed.

Summarizing, the prototype CUDA implementation, which uses only the slowest global memory and has no algorithmic optimizations applied, works over one order of magnitude faster than the standard CPU implementation.

4 Conclusions

We have demonstrated that the Graph of Cellular Automata model can be used as an interesting metaphor for modeling systems consisting of producing environment and transportation network. Previously this simulation method was successfully applied for modeling anastomosing river and vascular system developed by tumor-induced angiogenesis. We show that very similar schema can be applied for modeling proliferation of mycelium. This fact proves that GCA approach can be considered as an interesting universal tool for modeling complex biological phenomena connected with the phenomenon of directed growth. The potential of the GCA modeling scheme in simulating really large system was confirmed by its efficient implementation in GPGPU environment which gives at least one order of magnitude speedup in comparison to CPU implementation. Having in mind that not the full power of GPU was exploited a possible room for improvement still exists.

Acknowledgements. This research is financed by the Polish Ministry of Higher Education and Science, project N519 579338 and partially by AGH grant No. 11.11.120.777. We would like to thank Nvidia Company for support and donating Authors with hardware. The Authors are very grateful to Dr Shea Miller and Dr Margaret Balcerzak from Agriculture and Agri-food Canada for inspiration and providing us with the images of Fusarium Graminearum growth from fluorescent microscopy shown in Fig. 1.

References

1. Miller, S.S., Chabot, D.M.P., Ouellet, T., Harris, L.J., Fedak, G.: Use of a Fusarium graminearum strain transformed with green fluorescent protein to study infection in wheat (Triticum aestivum). Can. J. Plant Pathol. 26, 453–463 (2004)

2. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge University Press (1998)
3. Topa, P.: Network Systems Modelled by Complex Cellular Automata Paradigm. In: Salcido, A. (ed.) Cellular Automata - Simplicity Behind Complexity. InTech (2011) ISBN: 978-953-307-230-2
4. Wąs, J., Gudowski, B., Matuszyk, P.J.: Social Distances Model of Pedestrian Dynamics. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 492–501. Springer, Heidelberg (2006)
5. Pritsch, C., Muehlbauer, G.J., Bushnell, W.R., Somers, D.A., Vance, C.P.: Fungal development and induction of defense response genes during early infection of wheat spikes by Fusarium graminearum. Molecular plantmicrobe interactions MPMI 13, 159–169 (2000)
6. Gobron, S., Coltekin, A., Bonafos, H., Thalmann, D.: GPGPU Computation and Visualization of Three-dimensional Cellular Automata. The Visual Computer 27(1), 67–81 (2011)
7. CUDA Zone, http://www.nvidia.com/object/cuda_home_new.html
8. Khronos Group, http://www.khronos.org/opencl/
9. Boswell, G.P., Jacobs, H., Davidson, F.A., Gadd, G.M., Ritz, K.: A positive numerical scheme for a mixed-type partial differential equation model for fungal growth. Appl. Math. Comput. 138, 321–340 (2003)
10. Halley, J.M., Comins, H.N., Lawton, J.H., Hassell, M.P.: Competition, Succession and Pattern in Fungal Communities: Towards a Cellular Automaton Model. Oikos 70(3), 435–442 (1994)
11. Laszlo, J.A., Silman, R.W.: Cellular automata simulations of fungal growth on solid substrates. Biotechnology Advances 11(3), 621–633 (1993), Special Issue: Solid Substrate Fermentations
12. Boswell, G., Jacobs, H., Ritz, K., Gadd, G., Davidson, F.: The Development of Fungal Networks in Complex Environments. Bulletin of Mathematical Biology 69(2), 605–634 (2007)
13. Topa, P., Dzwinel, W., Yuen, D.: A multiscale cellular automata model for simulating complex transportation systems. International Journal of Modern Physics C 17(10), 1–23 (2006)
14. Topa, P.: Dynamically Reorganising Vascular Networks Modelled Using Cellular Automata Approach. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 494–499. Springer, Heidelberg (2008)
15. Topa, P.: Towards a Two-Scale Cellular Automata Model of Tumour-Induced Angiogenesis. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 337–346. Springer, Heidelberg (2006)
16. Harris, S.D.: Branching of fungal hyphae: regulation, mechanisms and comparison with other branching systems. Mycologia 100(6), 823–832 (2008)

DPD Model of Foraminiferal Chamber Formation: Simulation of Actin Meshwork – Plasma Membrane Interactions

Paweł Topa[1,2], Jarosław Tyszka[1], Samuel S. Bowser[3], and Jeffrey L. Travis[4]

[1] Institute of Geological Sciences, Polish Academy of Sciences, Biogeosystem Modeling Laboratory, Research Center in Kraków, ul. Senacka 1, PL-31002 Kraków, Poland
ndtyszka@cyf-kr.edu.pl
[2] AGH University of Science and Technology, al. Mickiewicza 30, Kraków, Poland
topa@agh.edu.pl
[3] Wadsworth Center, NYS Department of Health, USA
sam.bowser@bowserlab.org
[4] Department of Biological Sciences, University at Albany, USA
jefft@albany.edu

Abstract. Foraminifera are unicellular organisms which are widespread especially in marine environments. They produce protective shells (called tests) around their cell bodies, and these may be hardened by either secreted $CaCO_3$ or by the agglutination of sediment grains from the environment. Such mineralized shells readily fossilize, which makes them useful in paleo/environmental and related geological applications. The morphology of foraminiferal chambers emerges from a cascade of complex genetically-controlled processes ultimately controlled through the interactions among morphogenetic components. From studies on the morphogenesis and movements of foraminiferan pseudopodia, we presume that actin meshwork, microtubules, plasma membrane and their various associated proteins all contribute to chamber formation. Here, we apply dissipative particle dynamics (DPD) simulation techniques to model interactions between the plasma membrane and actin meshwork to test their role in the formation of cell body and test architecture. The present studies mark the first stage of "in silico" experiments aimed at developing an emergent model of foraminiferal chamber formation and shell morphogenesis.

Keywords: foraminifera, dissipative particle dynamics, cell physiology, complex fluids.

1 Introduction

Computer modeling can be a powerful approach to achieve a better understanding of complex dynamic systems. Building a computer model of any phenomenon requires in-depth analyses of all involved processes and serves as a virtual laboratory for experiments testing the effects of varying the magnitude of those

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 588–597, 2012.

processes on the emergent properties of the complex system. The strength of modeling is that it provides predicted observation of simulated processes that otherwise are out of spatial and/or temporal scales for any direct examination.

Following this approach, we have applied the dissipative particle dynamics (DPD) method ([1], [2], [3]) to simulate foraminiferal shell morphogenesis. Processes that "shape" foraminiferal shells are most likely similar in every cell in living organisms. The novelty of our approach is to gather selected components and phenomena into one computational framework. Our aim is to build a virtual tool that can give insight into processes that control the shapes of newly formed foraminiferal chambers.

Foraminifera are a familiar group of unicellular eukaryotic organisms (kingdom Protista) distributed worldwide mostly in marine and brackish habitats, although freshwater species are known to exist. These organisms produce protective shells (properly called "tests" because they are sometimes vested by protoplasm) to protect their relatively large and sensitive cell body and its nucleus. While much of the structural template for the test consists of organic material produced and fashioned by the cells, various taxonomic groups further fortify and harden the test with agglutinated mineral grains selected from the sediment, or by secreted $CaCO_3$. These mineralized shells are long lived and readily fossilize. As a result, foraminifera have left an impressive fossil record that shows very specific distributions of forms in both time and space that make them perfect targets for micropalaeontologic, biostratigraphic, paleoecologic, paleoceanographic, and paleoclimatologic studies (e.g. [4], [5]).

The shells of living and fossil foraminifera display a variety of test shapes and patterns (Fig. 1). They grow by iterative, successive formation of chambers attached to an embryonic shell, called the proloculus. The theoretical morphology of foraminiferal shells has been studied for more than 40 years ([6], [7], [8]), using models that construct the theoretical morphospace employing geometrical operations parameterized by ratios of translation and rotations. However, to date such models have not been able to simulate shells that reveal complex growth patterns, e.g., switching from one coiling mode to another during their virtual ontogenesis. Topa and Tyszka ([9], [10], [11]) demonstrated that the limited capacities of these models were a result of neglecting the role of the aperture in the process of chamber formation. The moving reference model [10] uses the aperture as a reference point with respect to which a new chamber is located (see Fig. 1). This assumption comes from the fact that the aperture (or multiple apertures) are shell openings responsible for communication between an internal cell body and external microhabitat explored by pseudopodial extensions called granuloreticulopodia ([5], [12], [13]). The same apertures define the initial position of successive chambers during shell growth.

Recent investigations aim at constructing a new emergent model of chamber formation (see [10]). Shapes of foraminiferal chambers emerge from the cascade of complex morphogenetic processes, basically controlled by genetic information. A new chamber follows a shape of the primary organic membrane ([14], [18]), and we presume that the primary organic membrane is shaped by cytoskeletal

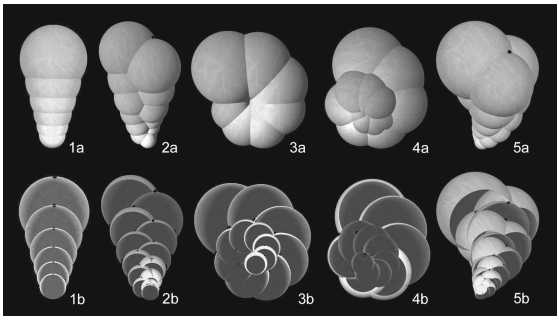


Fig. 1. Simulation of foraminiferal shells applying the moving reference model. Black dots represent foraminiferal apertures. Upper row presents external views of shells. Lower row shows cross-sections of these shells, showing their internal architecture. 1. uniserial chamber arrangement; 2: biserial; 3: planispiral; 4: low trochospiral (low helicoidal); 5: triserial high trochospiral. Such real shells range from 0.05 to 2 mm. Some complex foraminifera are much larger and reach up to several centimeters.

filaments and their associated motors and cross-linking proteins [15]. To gain insight into chamber morphogenesis, we need a technique that enables us to model the interactions between components responsible for cell architecture, such as a plasma membrane, sub-membrane actin filament meshwork, microtubules, and various associated proteins (e.g. [16], [17], [15]). The plasma membrane and its associated cytoplasm and extracellular domains can be classified as a complex fluid composed of liquid and suspended solid structures, making it amenable for study by Dissipative Particle Dynamics, one of the computer modeling methods frequently employed to study such fluids ([1], [2], [3]).

The DPD model consists of a set of particles that move off-lattice interacting with each other through the three type of forces: repulsive conservative force, dissipative force and stochastic force [1]. From a physical point of view, each particle can be regarded not as a single molecule but rather as a cluster of molecules. One of the most attractive features of the DPD technique is its enormous versatility in constructing simple models of complex fluids. For example, a simple Newtonian fluid can be made "complex" by adding additional interactions between fluid particles. Different particle-particle interactions can be introduced to model various other types of fluids. Polymers may be modeled as chains of molecules linked by springs.

The DPD method was applied to model selected phenomena associated with biological cells and tissues, as well as their components, such as plasma membranes, which can be approximated as complex fluids. For example, Basan et al. [19] proposed a DPD model for investigating the properties of tissue. Cells in this model were represented by DPD particles that adhere to each other, expand in volume, divide after reaching a specific size checkpoint, and undergo apoptosis at a constant rate, which ultimately leads to a steady-state homeostatic pressure in the tissue. Blood cells and the properties of blood flow have also been

investigated using the DPD approach. Fedosov et al. [21] modeled the behavior of red blood cell membranes, and Filipovic et al. [22] investigated various blood flow properties. Similarly, Boryczko and Dzwinel [23] tested the application of DPD to model blood flow in capillaries. Other investigations have been devoted to modeling of the cell membrane - lipid bilayer. Lipowsky et al. intensively studied the properties of lipid bilayers with various particles based on computational methods, including DPD ([24], [25], [20]). In their models, the lipid bilayer was formed from short chains of DPD particles; each chain consisted of hydrophilic "head" particles and hydrophobic "tail" particles. These chains self-organized into well-ordered bilayer structures due to strong repulsion parameters between water and hydrophobic particles. This approach was used in other models that investigated various properties of lipid bilayers and their behavior under various conditions ([26], [27], [28]).

Here, we present a preliminary DPD model of interactions between the actin meshwork, plasma membrane, and a solid wall as a part of processes most likely happening during foraminiferal chamber formation. The ultimate goal is to obtain a realistic model of the primary organic membrane shaped by pseudopodial cytoskeleton structures, such as an actin meshwork and microtubules.

2 Model

The DPD method was introduced to simulate the hydrodynamic behavior of complex fluids [1]. The elementary units of the DPD model are soft particles with mass m_0 and radius of interaction r_0. Their evolution is governed by Newton's equations of motion:

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{f}_i. \tag{1}$$

The force $\mathbf{f}_i$ acting on a particle is given by the sum of a conservative force, a dissipative force and a random force [2]:

$$\mathbf{f}_i = \sum_{j \neq i} (\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R) \tag{2}$$

A particle i interacts only with other particles j located at a distance less than a certain cutoff radius r_c.

The conservative force $\mathbf{F}_{ij}^C$ acts as soft repulsion force along the line of centers and it is defined as [2]:

$$\mathbf{F}_{ij}^C = a_{ij}(1 - \frac{|\mathbf{r}_{ij}|}{r_0})\hat{\mathbf{r}}_{ij} \tag{3}$$

where:

- a_{ij} is the maximum repulsion force between particle i and particle j,
- r_0 is the diameter of DPD particles,
- $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$,
- $\hat{\mathbf{r}}_{ij} = \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}$.

The dissipative force $\mathbf{F}_{ij}^{D}$ [2] removes energy from the system by decreasing the velocity if the two particles in relation to each other. The force does not affect particles that either move parallel to each other or overlap. By slowing down fast moving particles, the dissipative force makes the system more controllable and predictable.

$$\mathbf{F}_{ij}^{D} = -\gamma_{ij}(1 - \frac{|\mathbf{r}_{ij}|}{r_0})^2(\hat{\mathbf{r}}_{ij}\mathbf{v}_{ij})\hat{\mathbf{r}}_{ij} \tag{4}$$

where:

– γ_{ij} — friction coefficient that scales dissipative force.

The random force compensates for the loss of kinetic energy due to the dissipative force. It provides for the random motion of particles [2]:

$$\mathbf{F}_{ij}^{R} = \sqrt{2\gamma_{ij}k_BT}(1 - \frac{|\mathbf{r}_{ij}|}{r_0})\xi_{ij}\hat{\mathbf{r}}_{ij} \tag{5}$$

where:

– ξ_{ij} — a random variable with zero mean and unit variance,
– k_B — Boltzmann's constant,
– T — desired equilibrium temperature on the system in Kelvin.

In the DPD method, the particle radius r_c defines the length scales for the simulation. We assume that a single particle represents a volume equal to 3 water molecules [26]. Thus, the r_c is approximately equal to $0.7nm$. All other dimensions in our model are related to r_0, and later in the text, these values are expressed in r_0 units. Density of these particles are set to 3 per unit volume ($\rho r_c^3 = 3$). The time scale is much more difficult to evaluate in DPD. Usually it is individually calculated from other parameters as a diffusion rate [26] or thermal velocity [29].

Our model focuses on the influence of an underlying actin meshwork on plasma membrane behavior. At this stage the model is significantly simplified. The system consists of a partly opened box made of wall particles that cannot move (see Figure 2) and acts as a kind of "virtual shell". The box and space above is filled with liquid particles and actin filament - polymer chains. One of the walls of the box is made of a flexible membrane, which models the plasma membrane of a newly formed chamber. The membrane is made of particles that initially are organized in a regular mesh (see Figure 3B). Each membrane particle is connected with its four neighbors via discrete spring potentials, with connections that remain unchanged during the simulation. Additionally the bond angle potential is defined for every three membrane particles that form a straight line. These connections also remain unchanged during the simulation. The membrane sheet is attached to the hard walls also by spring potentials but with different k_1 parameters. All other types of particles interact with walls by applying the conservative repulsion force.

In this model, several types of particles are introduced (see Fig. 2):

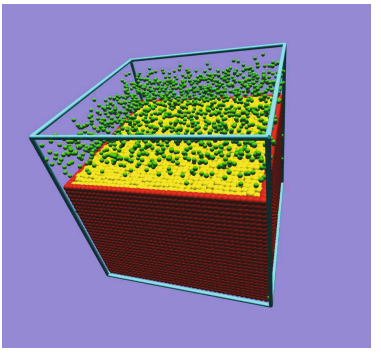


Fig. 2. System configuration tested during model development

- fluid (water) particles (F — vizualized with green color)
- actin particles (A — grey)
- membrane particles (M — yellow)
- wall particles (W — red)

The actin meshwork is composed of particle chains (see Figure 3A). Between neighboring particles in chains additional spring force is defined (see Figure 3C):

$$F_{ij}^{S} = -k_1(\mathbf{r}_{ij} - \mathbf{r}_0) \tag{6}$$

where:

- k_1 is the spring constant (as in Hook's law),
- $\mathbf{r}_0$ is the unstretched length between two neighbouring particles in chain.

In order to prevent chains from excessive bending we define an additional three-particle potential that straightens chains of particles, which follows the concept of bond angle potential introduced by Shilcock and Lipowsky [24] (see Figure 3 D).

$$U_B(i-1, i, i+1) = k_2(1 - cos(\phi - \phi_0)) \tag{7}$$

Various properties of particles in the model are defined through the conservative repulsion parameter a_{ij}. For water particles, we used a standard value of 25 (in units of $\frac{k_B T}{r_0}$) (following [3]). Actin particles repulse each other with a conservative force of the same value as the repulsive parameter $a_{ij}^{AA} = 25$. The spring constant k_1^A for chains of actin particles is set to 200, and the bending potential parameter k_2^A is set to 20. Membrane particles also interact each other with the repulsive parameter $a_{ij}^{MM} = 25$. Spring and bending parameters are: $k_1^M = 100$ and $k_2^M = 20$. Membrane particles that are initially located on the edge of the mesh are connected to the walls with spring constant $k_1^{MW} = 200$.

Spring and bending potentials are responsible for shapes and structures of actin chains and the membrane. Mutual interactions between fluid, actin meshwork, plasma membrane, and walls of the "virtual" chamber are controlled

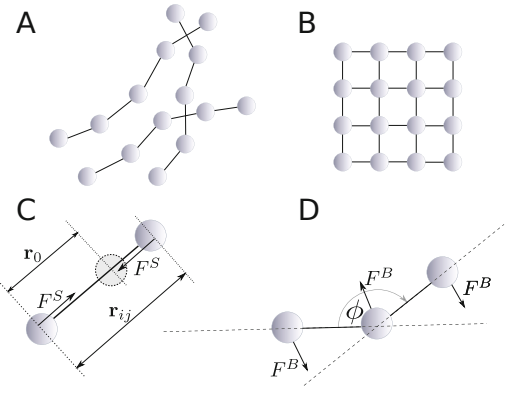


Fig. 3. Structures made of bounded particles and their potentials: A) actin meshwork modeled as chains of particles, B) regular mesh of membrane particles, C) two-body spring potential, D) three-body angle potential

mainly by the repulsive conservative force parameter a. In our experiments we focus on analyzing the behavior of the virtual actin meshwork for different sets of repulsive force parameters:

- a^{AM} — interaction between actin particles and membrane particles,
- a^{AW} — interaction between actin particles and wall particles,
- a^{AF} — interaction between actin particles and fluid particles.

In all simulations, the step of integration is set to typical value $\delta t = 0.04$ that follows Groot and Warren [3] suggestions. The simulations involve about 40 000 particles. For integration of Newton's equation of motion we use Velocity Verlet scheme. The model was implemented in C++ language for Linux platform. Algorithms are parallelized with OpenMP API. Results are visualized with OpenGL library.

3 Results

Figure 4 presents sample results of our simulations. Fluid and wall particles are removed for clarity. Parameters of conservative force was set to the following values: Fig.4**A** $a^{AW} = 6, a^{AM} = 6, a^{AF} = 12$, Fig.4**B** — $a^{AW} = 12, a^{AM} = 6, a^{AF} = 12$. Actin filaments fill all available space and act on the membrane so as to deform it. In both cases, it can be observed that single actin filaments have penetrated through the membrane, due to fact that actin and membrane particles are "soft" particles that can interpenetrate. This effect can be controlled by modifying the conservative force parameter.

Figure 5 presents results of "in silico" experiments conducted on a model actin meshwork. Simulations were performed for different sets of the repulsive force parameter: Fig. 5A: $a^{AW} = 50, a^{AM} = 2, a^{AF} = 25$ and Fig. 5B:

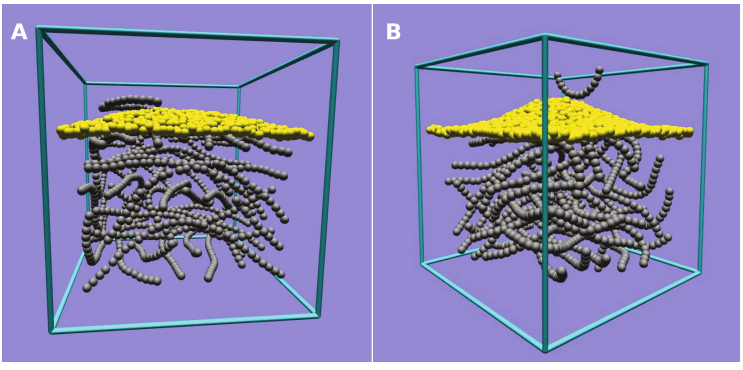


Fig. 4. Sample results of simulation with actin meshwork (grey) influencing membrane shape (yellow), see text for details

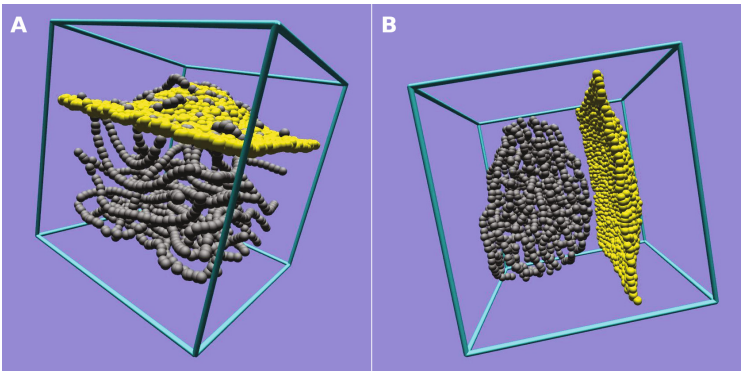


Fig. 5. Sample results of "in silico" experiments (see text for details)

$a^{AW} = 25, a^{AM} = 2, a^{AF} = 50$. We observed different behaviors of the actin meshwork and plasma membrane. Fig. 5A shows attachment of actin filaments to the membrane. The meshwork is also strongly repulsed from the walls. The simulation in Fig 5B shows a strong repulsion between actin and fluid particles compressing the meshwork into a globular tangle of filaments.

4 Conclusions

We present a preliminary version of our DPD model of foraminiferal chamber morphogenesis. Our model, at this stage of its development, focuses on simulating the actin filament meshwork. We represented it by a set of particle chains connected with two kind of potentials: spring and bending. Actin filaments

interact with the membrane which is made of particles also connected with spring and bending potentials. We are searching for proper behavior of the modeled system by modifying parameters of the conservative force. Our future works will focus on:

- improving the membrane model by incorporating the lipid bilayer model defined by Lipovsky et al. [24];
- introducing microtubules as a main morphogenetic component;
- modeling mechanical properties of membrane in contact with microtubules.

Acknowledgements. We thank Dirk de Beer, Jelle Bijma, Antje Funcke, Martin Glas, Nina Keul, Gerald Langer, Gernot Nehrke, Lennart J. de Nooijer, Jan Pawlowski, Lubos Polerecky and Silke Thoms for fruitful discussions. We are especially grateful to the late Prof. Lukas Hottinger for sharing his knowledge and scientific enthusiasm. This research is supported by the Kosciuszko Foundation and the Polish Ministry of Science and Higher Education, project no. N N307057334.

References

1. Hoogerbrugge, P.J., Koelman, J.M.V.A.: Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. Europhysics Letters 19(3), 155–160 (1992)
2. Español, P., Warren, P.B.: Statistical-mechanics of dissipative particle dynamics. Europhysics Letters 30(4), 191–196 (1995)
3. Groot, R.D., Warren, P.B.: Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation. The Journal of Chemical Physics 107(11), 4423–4435 (1997)
4. Culver, S.J.: Foraminifera. In: Lipps, J.H. (ed.) Fossil Prokaryotes and Protists, pp. 203–247. Blackwell, Boston (1993)
5. Sen Gupta, B.K.: Modern Foraminifera. Kluwer, Dordrecht (1999)
6. Berger, W.H.: Planktonic foraminifera; basic morphology and ecologic implications. Journal of Paleontology 43(6), 1369–1383 (1969)
7. Signes, M., Bijma, J., Hemleben, C., Ott, R.: A model for planktic foraminiferal shell growth. Paleobiology 19, 71–91 (1993)
8. Raup, D.M.: Geometric analysis of shell coiling: General problems. Journal of Paleontology 40, 1178–1190 (1966)
9. Topa, P., Tyszka, J.: Local Minimization Paradigm in Numerical Modelling of Foraminiferal Shells. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J., Hoekstra, A.G. (eds.) ICCS 2002, Part I. LNCS, vol. 2329, pp. 97–106. Springer, Heidelberg (2002)
10. Tyszka, J., Topa, P.: A new approach to modeling of foraminiferal shells. Paleobiology 31(3), 526–541 (2005)
11. Tyszka, J.: Morphospace of foraminiferal shells: results from the moving reference model. Lethaia 39(1), 1–12 (2006)
12. Travis, J.L., Bowser, S.S.: Microtubule-dependent reticulopodial motility: Is there a role for actin? Cell Motility and the Cytoskeleton 6(2), 146–152 (1986)
13. Bowser, S.S., Travis, J.L.: Reticulopodia: structure and behavioral basis for the suprageneric placement of granuloreticulosean protists. Journal of Foraminiferal Research 32, 440–447 (2002)

14. Bé, A.W.H., Hemleben, C., Anderson, O.R., Spindler, M.: Chamber formation in planktonic foraminifera. Micropateontology 25, 294–307 (1979)
15. Tyszka, J., Bowser, S.S., Travis, J.L., Topa, P.: Self-organization of foraminiferal morphogenesis. In: International Symposium on Foraminifera FORAMS 2010, Bonn 5-10.10.2010, p. 192 (2010)
16. Etienne-Manneville, S., Hall, A.: Rho GTPases in cell biology. Nature 420(6916), 629–635 (2002)
17. Laplante, C., Nilson, L.: Differential expression of the adhesion molecule Echinoid drives epithelial morphogenesis in *Drosophila*. Development 133, 3255–3264 (2006)
18. Weiner, S., Erez, J.: Organic matrix of the shell of the foraminifer, *Heterostegina depressa*. Journal of Foraminiferal Research 14(3), 206–212 (1984)
19. Basan, M., Prost, J., Joanny, J.F., Elgeti, J.: Dissipative particle dynamics simulations for biological tissues: rheology and competition. Physical Biology 8(2), 26014 (2011)
20. Grafmüller, A., Shillcock, J.C., Lipowsky, R.: Dissipative particle dynamics of tension-induced membrane fusion. Molecular Simulation 35, 554–560 (2009)
21. Fedosov, D.A., Caswell, B., Karniadakis, G.E.: Dissipative Particle Dynamics Modeling of Red Blood Cells. In: Pozrikidis, C. (ed.) Computational Hydrodynamics of Capsules and Biological Cells, pp. 183–218. CRC Press (2010)
22. Filipovic, N., Ravnic, D., Kojic, M., Mentzer, S.J., Haber, S., Tsuda, A.: Interactions of blood cell constituents: Experimental investigation and computational modeling by discrete particle dynamics algorithm. Microvascular Research 75(2), 279–284 (2008)
23. Dzwinel, W., Boryczko, K., Yuen, D.A.: A Discrete-Particle Model of blood dynamics in capillary vessels. Journal of Colloid and Interface Science 258(1), 163–173 (2003)
24. Shillcock, J.C., Lipowsky, R.: Equilibrium structure and lateral stress distribution of amphiphilic bilayers from dissipative particle dynamics simulations. Journal of Chemical Physics 117(10) (2002)
25. Lianghui, G., Shillcock, J.C., Lipowsky, R.: Improved dissipative particle dynamics simulations of lipid bilayers. Journal of Chemical Physics 126(015101) (2007)
26. Groot, R.D., Rabone, K.L.: Mesoscopic simulation of cell membrane damage, morphology change and rupture by nonionic surfactants. Biophysical Journal 81(2), 725–736 (2001)
27. Yamamoto, S., Maruyama, Y., Hyodo, S.: Dissipative particle dynamics study of spontaneous vesicle formation of amphiphilic molecules. Journal of Chemical Physics 116(13), 5842 (2002)
28. Ganzenmüller, G.C., Hiermaier, S., Steinhauser, M.O.: Shock-wave induced damage in lipid bilayers: a dissipative particle dynamics simulation study. Soft Matter 7, 4307–4317 (2011)
29. Symeonidis, V., Karniadakis, G.E., Caswell, B.: Dissipative particle dynamics simulations of polymer chains: scaling laws and shearing response compared to DNA experiments. Phys. Rev. Lett. 95(7) (2005)

A Discrete Simulation Model for Traffic Including Bicycles on Urban Networks, Applied to Intersection of Two One-Way Streets

Jelena Vasic and Heather J. Ruskin

School of Computing, Dublin City University, Glasnevin, Dublin 9, Ireland
jelena.vasic@computing.dcu.ie

Abstract. The contemporary problem of 'greening' urban transport, in its complexity, calls for multi-faceted solutions, including a range of inputs, from theoretical modelling to resource planning and monitoring. This paper offers a contribution to scenario assessment, through a computational model of heterogeneous urban traffic flows. Built on a cellular-automaton framework, it considers, in particular, the inclusion of pedal-bicycles in the specific case of a city with an incomplete dedicated bicycle lane network, where close road sharing takes place between motorised and non-motorised vehicles. Generic spatial and behaviour components of the model have been defined and a simulation framework built based on those. Application of this framework is demonstrated for the case of an intersection of two one-way streets.

Keywords: traffic flow modelling, cellular automata, bicycles.

1 Introduction

In recent times, the push towards transport sustainability, within the wider background of today's environmental concerns, has brought about increased involvement with non-motorised modes of transport, by policy-makers, researchers and participants alike [13]. The health and social benefits of these are also significant [5]. In this context, understanding the physical properties of traffic flow including bicycles is of interest, both in support of promoting and planning for integration in developed countries and for the alleviation of congestion and improvement in safety in developing countries [15]. This paper describes a cellular automaton-based modelling method for modelling heterogeneous traffic on arbitrary network elements and its application to a particular scenario, with results that illustrate how the model can be used. The remainder of this section gives an overview of existing models for mixed traffic including bicycles. Section 2 is dedicated to summarising the spatial modelling method used and the corresponding movement rules. Following that, Section 3 describes how the single-lane one-way road intersection model is defined using that method. Simulation results are presented in Section 4. Section 5 concludes the paper, by summarising the results and placing them in the context of future work.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 598–607, 2012.

Since the seminal publications on the application of cellular automata (CA) to traffic flow modelling in the early nineties [12,1], that type of model has been widely used in the context of transportation science. Traffic heterogeneity, which is of interest to us, is found in its simplest CA form in models where the (one-cell-wide) vehicles can have different lengths (expressed in cells). In models that employ this technique, the mix represented is always that of smaller and larger motorised vehicles, as is the case in the network model by [3] or in [2] and [4], where particular rules are employed to simulate interactions within a mix of trucks and cars, at an intersection and a roundabout, respectively. A mix of vehicles with different widths, which generally contains non-motorised vehicles, however, presents a modelling problem that cannot be solved by simple use of one-dimensional CA models.

Models for mixed motorised and non-motorised flows must be appropriate for the particular traffic mix and the manner in which the flows are composed. Thus, models of mixed traffic in India must represent flows arising from an absence of lateral positioning rules, i.e. flows that are "homogeneously heterogeneous"; models of mixed bicycle and motorised traffic in China are required to reflect flows on roads with dedicated bicycle lanes that accommodate several bicycles travelling abreast alongside motorised traffic lanes; and models for Dublin have to deal with predominantly single-file bicycle traffic sharing mostly narrow lanes with motor vehicles, in the absence of bicycle-specific controls. Accordingly, the models reviewed in [9], including [7], aimed to reproduce the broadly heterogeneous traffic typical of Indian cities, mostly through spatially continuous simulation. [6] and [11] developed CA models with multiple cell occupancy, reflecting size and shape of different vehicles, for traffic with those same characteristics, on a stretch of road. The Chinese conditions are tackled in a model by [8], where the idea to use a discrete form of the Burgers equation as a basis for CA update rules [14], is applied to bicycle traffic, for which it is quite suitable, since it allows multiple occupancy of cells. [10] use the same model for the study of interactions between car and bicycle flows at an intersection. The car-following model described in [17] allows for interactions between motorised and non-motorised vehicles in two dimensions and includes impact of other vehicles positioned laterally and behind, when determining the behaviour of any individual unit.

2 General Model

In [16] we introduced a general *spatial model* definition method, which facilitates the transposition of natural geometric representations of network elements or sections into abstract representations in a systematic way. Figure 1 shows examples of all the constructs that are used with this method. Their descriptions follow.

1. The one-dimensional cellular automaton space (*OCAS*) is the basic building block. A vehicle on a network constructed using this method is always moving along an OCAS[1]. An OCAS consists of cells that are all the same size, but the

[1] In the case of lateral movements, e.g. lane-changing, the vehicle would 'hop' into another OCAS. While considered in the general model, such movements are not dealt with here.

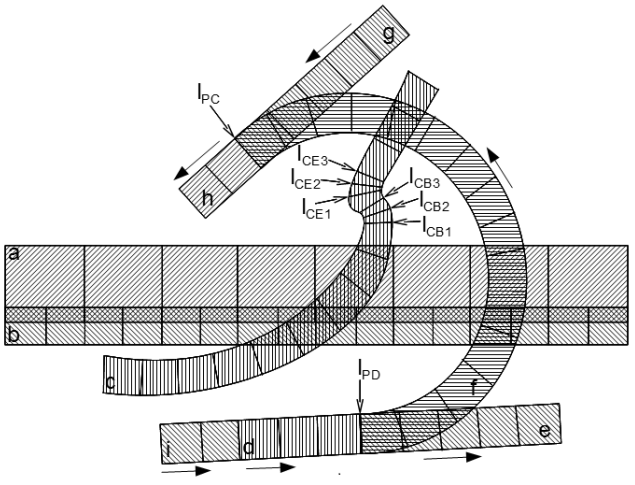


Fig. 1. Examples of spatial model constructs

cells belonging to one OCAS can be of a different size to the cells belonging to another OCAS (e.g. OCAS *a* and OCAS *b* in Figure 1). Differently sized vehicles are accommodated by OCASes with different cell sizes.

2. An OCAS can connect to another OCAS with the same cell size, so that a vehicle moves from the last cell of the first OCAS onto the first cell of the second OCAS (e.g. in Figure 1 OCAS *i* is connected to OCAS *d*).If an OCAS connects to more than one other OCAS (as is the case with OCAS *d* in Figure 1, which connects to OCASes *f* and *e*), this causes a *divergence*. Similarly, more than one OCAS connecting to a single OCAS form a *convergence* (OCASes *f* and *g* connecting to OCAS *h* is an example).
3. OCASes can run across each other, or intersect (for example, OCAS *c* and OCAS *f* intersect in Figure 1).
4. OCASes can belong to the same road (OCASes *a* and *b* in Figure 1 belong to the same road). OCASes on the same road may overlap, as in the last example, which means that two vehicles, if each is moving on one of the OCASes, cannot pass each other.
5. One final construct allowed by the modelling method is overlap between cells within an OCAS itself. This serves the purpose of imposing a slow-down on a vehicle passing through the overlapping cells (an example is shown in Figure 1, in OCAS *c*: the tree cells starting, respectively, at lines l_{CB1}, l_{CB2} and l_{CB3} and ending, respectively, at lines l_{CE1}, l_{CE2} and l_{CE3} all overlap with each other[2].

The *behaviour model* is based on the Nagel-Schreckenberg (N-S) update rules [12]. These can be formulated as follows:

[2] The example of overlap within an OCAS in Figure 1 is for demonstration purposes, as it is needed only where it models delay in areas of interaction with other OCASes.

1. Acceleration: if $v_i < v_{\mathrm{MAX}}$ and $v_i < d_i$, $v_i \to v_i + 1$
2. Slowing: if $d_i < v_i$, $v_i \to d_i$
3. Randomisation: with probability p_{R}, $v_i \to v_i - 1$
4. Vehicle motion: each vehicle is advanced v_i cells

where i is a vehicle's identifying index, v_i is the i^{th} vehicle's velocity (in cells per unit of time), d_i is the number of free cells to the nearest vehicle ahead of the i^{th} vehicle (in cells) and v_{MAX} is the maximal velocity. The first three rules update the velocity (e.g. $v_i \to v_i + 1$ means that velocity v_i is increased by 1). The first two are based around velocity limit $\min(v_{\mathrm{MAX}}, d_i)$ imposed jointly by the maximal velocity and the distance to the vehicle ahead, while rule 3 introduces stochasticity. The fourth rule prescribes that the i^{th} vehicle be moved along the CA space by v_i cells in the appropriate direction. The updates are synchronous, which means that rules 1-3 are applied to all vehicles in the system, followed by movement of all vehicles in the system, at each time step.

Our model builds on these rules, extending them in a number of ways:

- The rules are applied both to **bicycles and cars**. Each type of vehicle has a separate v_{MAX} value.
- Instead of counting the free cells to the nearest vehicle ahead, the model counts the number of **un-impinged** cells ahead. An impinged cell is that for which any overlapping cell (including itself) is occupied. The implementation represents overlaps between cells with bits, which makes the checking of impingement efficient.
- Another three factors are added to contribute to the **velocity limitation**.
  1. proximity of an intersection at which it is going to turn
  2. proximity of a conflict point (either an OCAS merge or an intersection of OCASes)
  3. (in the case of cars only) the longitudinal proximity of a bicycle in an OCAS on the same road and adjoining the car's OCAS

  The three factors are quantified in Table 1.

Table 1. Velocity limits on approach to turn/conflict/bicycle, chosen so as to allow vehicles to reach the velocity of 1 at the last cell before turn/bicycle and 0 before unresolved conflict, while decelerating, at most, by 1. The table is for maximal velocity, v_{MAX}, of 3 for cars and 2 for bicycles. The numbers in the first row represent the number of steps ahead that would cause entry into turn or conflict on bring a car beside a bicycle.

	6	**5**	**4**	**3**	**2**	**1**	**0**
Max velocity of turning bicycle	-	-	-	-	1	1	-
Max velocity of turning car	-	2	2	2	1	1	-
Max velocity of bicycle at conflict	-	-	-	1	1	0	-
Max velocity of car at conflict	2	2	2	1	1	0	-
Max velocity of car near bicycle	-	2	2	2	1	1	1

– For the **navigation** of divergences vehicles are simply assigned *probability of turning* values. As part of the specification of a spatial element, conflicts are characterised by a resolution rule, where one OCAS has priority and the other must give way. The model applies a *'soft yield'* to all conflict situations where a vehicle has to give way. This means that it inspects for any arriving vehicles in the conflicting OCAS and if it deduces, based on the other vehicle's velocity, distance and the rules for vehicle behaviour, that the other vehicle cannot, in the next time step, reach a cell that overlaps with it's own path, the inspecting vehicle advances. The inspecting vehicle only avoids crashes but does not attempt to clear the intersection quickly enough to ensure it is not obstructing other vehicles.

Armed with this set of rules for traversing any combination of spatial items defined using the method described earlier, vehicles do not need to learn any additional behaviour, even for the simulation of traffic on arbitrarily complex networks.

3 Intersection of Two Single-Lane Roads

Figure 2 shows the geometry of two spatial elements, a straight road stretch and an intersection of two one-way roads, and how these have been reduced to a number of representative OCASes of interest. From the picture, the following information was extracted to form the abstract model of the space. The abstract model, with information types numbered to match the corresponding spatial model construct descriptions in Section 2, follows.

1. List of OCASes deduced from the pictures. For stretch of road in Figure 2a: car and bicycle OCAS, CSR{50} and BSR{100}. For intersection in Figure 2b: CSN{2} (car south-north[3]), CSW{4O} (car south-west), BSN{4} (bicycle south-north), BSW{4O} (bicycle south-west), CEW{2} (car east-west), CEN{4O} (car east-north), BEW{4} (bicycle east-west), BEN{4} (bicycle east-north). The number of cells in the OCAS is given in curly brackets, followed by an O if some of the cells in the OCAS overlap. Bicycle OCAS length is half the length of a car OCAS; bicycle OCAS width is half the width of a car OCAS.
2. List of all existing and potential OCAS connections, which implicitly provides information on convergences and divergences. These do not occur in the road stretch, while in the intersection they are found as follows: an external bicycle OCAS may be connected to OCAS pair (BSN, BSW) or to (BEW, BEN), while an external car OCAS may be connected to OCAS pair (CSN, CSW) or to (CEW, CEN), each of those four cases forming a divergence; OCAS pairs (BSN, BEN), (BSW, BEW), (CSN, CEN) and (CSW, CEW) may each be connected to an external OCAS of the appropriate vehicle type, to form a convergence.

[3] These directions are taken to be north upwards on the page etc.

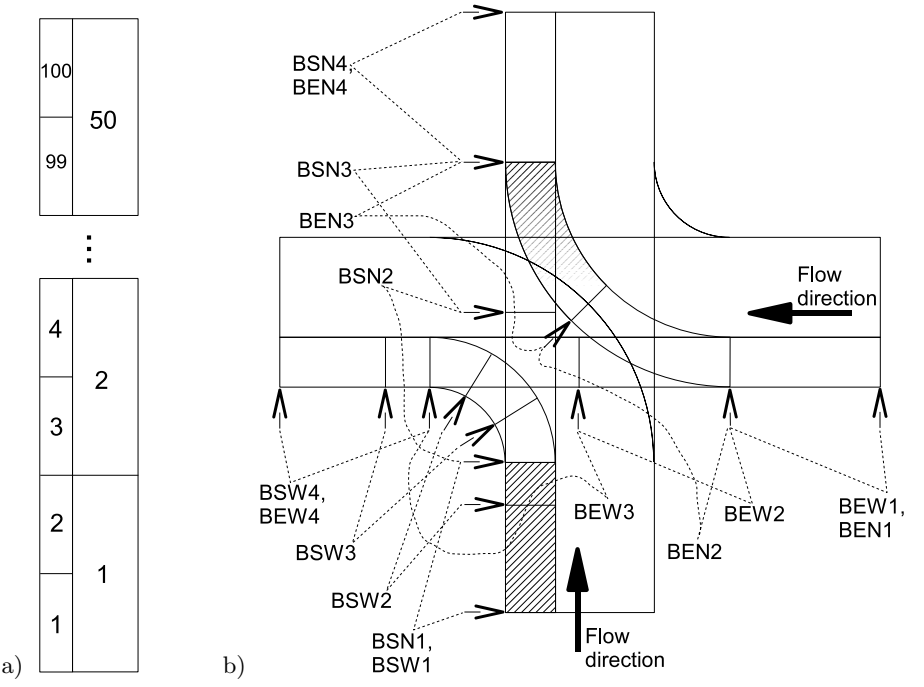


Fig. 2. Spatial model of (a) a road stretch and (b) an intersection of two one-way streets. The road stretch consists of two OCASes: as bicycle one 100 cells long and a car one 50 cells long, which run alongside each other. The element consists of four pairs of OCASes: south-north (SN), south-west (SW), east-west (EW) and east-north (EN). Each pair includes a narrower bicycle OCAS to the left (when viewing in the direction of movement) and a car wider OCAS to the right, to reflect the positional discipline with which cyclists share roads in cities like Dublin. The division of only bicycle OCASes into cells is shown, to avoid clutter in the picture. The bicycle SW OCAS (BSW) consists of four overlapping cells (BSW1, BSW2 etc.) and so does the bicycle EN (BEN) OCAS. Bicycle SN and EW OCASes have 4 non-overlapping cells each. The car OCASes, although this is not shown in the picture, SN and EW have 2 non-overlapping cells each, while SW and EN have 4 overlapping cells each.

3. Information on all the conflicts in the spatial element. An example of such a conflict is that occurring between the car SW OCAS and the bicycle SN OCAS in the intersection. This particular conflict can be described fully with information about the affected cell ranges: cells 1-4 of the CSW OCAS and 2-3 of the BSN OCAS; and about direction of conflict: from the left for CSW OCAS and from the right for BSN OCAS. There are 13 conflicts in the intersection and none in the road stretch.
4. Information on any bicycle OCAS adjoining a car OCAS: adjoining pairs are (BSR, CSR), (BSN, CSN), (BSW, CSW), (BEW, CEW) and (BEN, CEN).
5. An *impingement table*, recording all overlaps between cells, transposed from the graphical representation of the OCASes. The entries are binary, indicating

overlap/no overlap between each pair of cells. The impingement table for the stretch of road consists of zeroes only.

This information, extracted from the geometric representation of a spatial element, is used by the vehicles in applying the update rules.

4 Simulation Scenario and Results

The simulation scenario spatial model is shown in Figure 3. It consists of an intersection element and four straight road elements, like those in Figure 2.

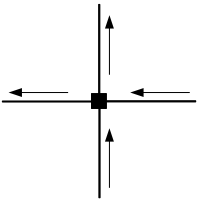


Fig. 3. Schematic representation of simulation scenario space. The intersection from Figure 2b is connected to 4 stretches of road from Figure 2a to form the shape of a '+'. The arrows show the direction of flow for each stretch of road.

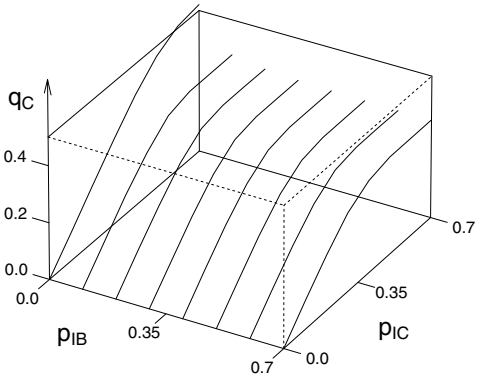


Fig. 4. CSN (car south-north) flow as a function of car insertion probability, for several values of bicycle insertion probability. All $p_{\mathrm{I}} = 0$ in EW direction and all $p_{\mathrm{T}} = 0$.

Maximal velocities of 3 and 2 for cars and bicycles, respectively, were used. With Nagel and Scheckenberg [12] real-life value assignments of 7.5m cell lengths and 1s time steps, this corresponds to 81km/h and 27km/h, which can be considered reasonable upper limits for an urban setting. The randomisation parameter used is 0.1. Insertion of vehicles at the beginning of the east and south roads, takes place at each time step, before rule application, with a certain probability of insertion for each OCAS. A new vehicle is placed on the initial stretch of road at position $v_{\mathrm{MAX}} - 1$ or the farthest unimpinged cell, whichever is lesser on the south and east end of the simulation space. Simulations were run for two priority cases: one where priority is given to vehicles on the east-west road and one where priority is given to vehicles on the south-north road.

As simulations have shown that flow values do not increase for values of insertion probability greater than 0.7 hence the diagrams do not include higher insertion probability values.

Figure 4 shows the car flow (number of cars that pass a certain point in the road per unit of time) values for CSN OCAS, as a function of car SN insertion

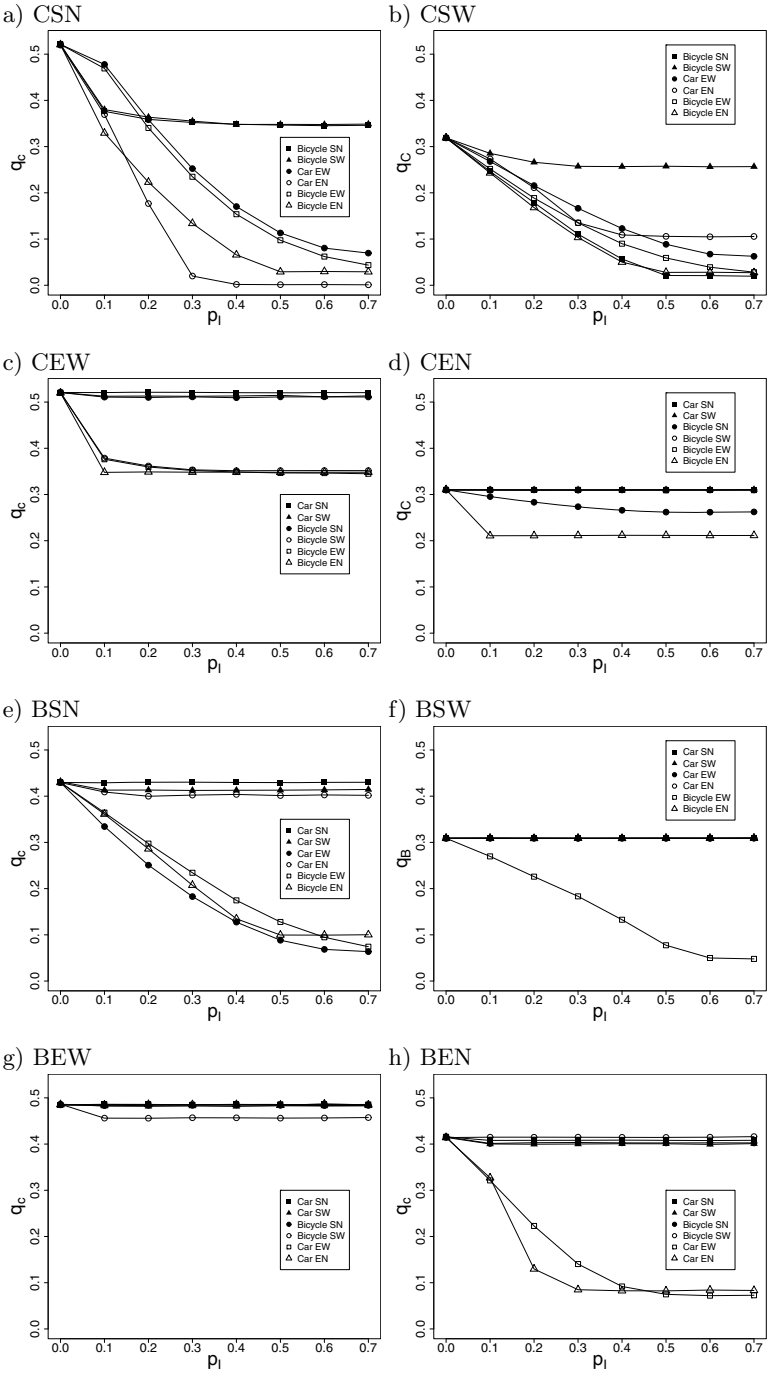


Fig. 5. Flow capacities as a function of other flows. The capacities are shown for each OCAS in the intersection in the case of priority given to the east-to-west road.

probabilities (probability of vehicle insertion onto the CSN OCAS of the south road section), for a selection of bicycle input probabilities for the BSN OCAS. The CSN OCAS capacity, which is the CSN OCAS flow value at insertion probability of 0.7, falls with the presence of bicycles on the BSN OCAS, since cars slow down in the presence of bicycles, according to the behaviour rules.

The capacity information has been extracted from simulation results for each OCAS in the intersection, as affected by single flows on other OCASes in the intersection. Figure 5 shows the results of this extraction for the case of east-to-west flow priority. The dramatic effect of most of the other flows on all the non-priority OCASes is immediately noticeable. Also, the BEN flow is negatively affected by the CEW and CEN flows, even though it is on the priority road, because "local" priority lies with cars sharing the same road.

5 Conclusion

The method developed for transferring the geometry of modelled infrastructure into an abstract representation is designed to facilitate scalability of the represented network and the systematic incorporation of new network elements into the model for heterogeneous traffic. A two one-way road intersection is modelled using this method as an example of the model's application and to demonstrate the utility of the model in reproducing qualitative relationships between the many flows of a network segment.

Further work will involve the development of a catalogue of network elements and investigation of interactions between bicycle and motorised flows on a large network. Because comprehensive real data, and especially single-vehicle data, for heterogeneous traffic and inter-modal interactions in a network are not readily available, initial quantitative assessment of results will be based on sensitivity analysis of network simulation result ranges and emergent properties respect to un-doubtable relations such as the fact that bicycles move more slowly than cars, that cars slow down (rather than speed up) in the presence of bicycles on a narrow shared street and the patterns of cross flow found on real streets. This, in combination with flow data, which is available for some scenarios, may provide a basis for the model's successful calibration. If it proves to be insufficient, data collection by the authors may be required.

Acknowledgement. This work is funded by the Irish Research Council for Science, Engineering and Technology (IRCSET), through an 'Embark Initiative' postgraduate scholarship, addressing the 'greening' of city transport.

References

1. Biham, O., Middleton, A., Levine, D.: Self-organization and a dynamical transition in traffic-flow models. Phys. Rev. A 46, R6124–R6127 (1992)

2. Deo, P., Ruskin, H.J.: Comparison of Homogeneous and Heterogeneous Motorised Traffic at Signalised and Two-Way Stop Control Single Lane Intersection. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3980, pp. 622–632. Springer, Heidelberg (2006)
3. Esser, J., Schreckenberg, M.: Microscopic simulation of urban traffic based on cellular automata. Int. J. Mod. Phys. C 8, 1025–1036 (1997)
4. Feng, Y., Liu, Y., Deo, P., Ruskin, H.: Heterogeneous traffic flow model for a two-lane roundabout and controlled intersection. Int. J. Mod. Phys. C 18, 107–117 (2007)
5. Geurs, K.T., Boon, W., Van Wee, B.: Social impacts of transport: Literature review and the state of the practice of transport appraisal in the netherlands and the united kingdom. Transport Rev. 29, 69–90 (2009)
6. Gundaliya, P., Mathew, T., Dhingra, S.: Heterogeneous traffic flow modelling for an arterial using grid based approach. J. Adv. Transport 42, 467–491 (2008)
7. Hossain, M., McDonald, M.: Modelling of traffic operations in urban networks of developing countries: a computer aided simulation approach. Computers, Environment and Urban Systems 22, 465–483 (1998)
8. Jiang, R., Jia, B., Wu, Q.: Stochastic multi-value cellular automata models for bicycle flow. Journal of Physics A - Mathematical and General 37, 2063 (2004)
9. Khan, S., Maini, P.: Modeling heterogeneous traffic flow. Transport Res. Rec. 1678, 234–241 (1999)
10. Li, X., Gao, Z., Jia, B., Zhao, X.: Modeling the interaction between motorized vehicle and bicycle by using cellular automata model. Int. J. Mod. Phys. C 20, 209–222 (2009)
11. Mallikarjuna, C., Rao, K.: Cellular automata model for heterogeneous traffic. J. Adv. Transport 43, 321–345 (2009)
12. Nagel, K., Schreckenberg, M.: A cellular automaton model for freeway traffic. J. Phys. 12, 2221–2229 (1992)
13. National Sustainable Transport Office, D.o.T.: Ireland's first national cycle policy framework 2009-2020. Tech. rep., Department of Transport (2009)
14. Nishinari, K., Takahashi, D.: Analytical properties of ultradiscrete burgers equation and rule-184 cellular automaton. Journal of Physics A - Mathematical and General 31, 5439 (1998)
15. Tiwari, G.: Traffic Flow and Safety: Need for New Models in Heterogenous Traffic. In: Injury Prevention and Control, pp. 71–88. Taylor & Francis (2001)
16. Vasic, J., Ruskin, H.J.: A Discrete Flow Simulation Model for Urban Road Networks, with Application to Combined Car and Single-File Bicycle Traffic. In: Murgante, B., Gervasi, O., Iglesias, A., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2011, Part I. LNCS, vol. 6782, pp. 602–614. Springer, Heidelberg (2011)
17. Xie, D., Gao, Z., Zhao, X., Li, K.: Characteristics of mixed traffic flow with non-motorized vehicles and motorized vehicles at an unsignalized intersection. Physica A 388, 2041–2050 (2009)

Lightweight Information Flow Control for Web Services

Bartosz Brodecki, Michał Kalewski, Piotr Sasak, and Michał Szychowiak

Poznań University of Technology
Piotrowo 2, 60-965 Poznań, Poland
{bbrodecki,mkalewski,psasak,mszychowiak}@cs.put.poznan.pl

Abstract. This paper presents a concept of incorporating information flow control (IFC) mechanisms into service-oriented systems. As opposed to existing IFC proposals, commonly imposing requirements hard or impossible to achieve in service-oriented environments (such as analysis of the application code), our solution fully complies with the Service Oriented Architecture (SOA) model. We present how IFC can be managed in an SOA system by using ORCA security policy language. We also describe two possible implementations of such SOA-specific IFC mechanisms using cryptographic keys and poly-instantiated web services.

1 Introduction

Service Oriented Architecture (SOA [8]) is today a well-known paradigm for developing large scale distributed systems which can run complex computations and take requests from many users distributed all over the world. One of the key problems in such systems concerns security issues. However, most of the security solutions provided today for the SOA systems only take access control issues into consideration. The approaches used for modeling security restrictions suffer from lack of information dissemination control after access to this information has been once authorized. Allowing a service uncontrolled use of any received confidential information (e.g. when further invoking other services) causes a serious risk of unauthorized information dissemination. Thus, information flow control (IFC [4]) mechanisms are an imperative security requirement which must be addressed where any sensitive information is processed in an SOA environment. Informally, IFC requires the system to ensure noninterference [5] which identifies the authorized information flow in the following way: low security level outputs of a program may not be directly affected by its high security level (confidential) inputs. Low level data can never be influenced by high level data in such a way that someone is able to determine the high level data from the low level output. The real world case of using IFC for securing information often arises in the context of HIS (Health Information Systems). Let us consider a hospital using SOA-based software to manage his day-to-day activities and patients' medical data. This information is often highly confidential and must be released only to authorized medical personnel. Several hospital departments, such as cardiology,

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 608–617, 2012.

accounting or main pharmacy, share different information regarding patients and their Electronic Health Records. However, specific information categories must be released only to the authorized departments and under strict control, e.g. as personal treatment data must not be available to the accounting department. It is the responsibility of IFC to ensure this.

One can observe that the IFC idea of controlling access to data is somehow similar to goals of Digital Right Management (DRM) technologies and languages such as XrML [3]. They are used to control access to various types of digital media. To some extent, both DRM and IFC have similar goals. However, DRM focus is on copyright protection, while IFC is focused on controlling information propagation across a system. Thus, in practice, IFC requires specific approaches and solutions.

There is some ongoing research on how to implant IFC in service-oriented systems. The main difficulty arises from the fact that the knowledge necessary to precisely analyze and understand the exact information flow can be derived from the source code only [10]. Knowing the source code, we can analyze what data is received at the input, how this data is processed and at which output channel it is delivered. This effectively gives us a chance to recognize and stop unauthorized information flow. Source code analysis, however, is known to be difficult to perform, time consuming and generating additional costs [15]. Moreover, it is impractical for service-oriented systems, since the SOA implementations offer full autonomy of services possibly originating from third-party providers. Thus, it is hard or impossible to externally enforce IFC mechanisms inside the application code of provided services. Moreover, there is usually no direct way to verify the presence of IFC mechanisms inside the service.

The *language-based* IFC approach focuses on using special languages for defining flow control restrictions. For example, the approach presented in [7] offers direct control on how information is processed using static analysis of information flow through Jif (Java information flow). Jif is used to certify programs to permit only authorized information flows. There are also proposals of using type calculus for IFC [12,13]. The main drawback of all these approaches is the requirement for source code developers to use specific languages.

Another approach, the *trust-based* [11,16], is more suitable for service-oriented systems. This approach assumes assigning a *trust level* to applications. This level reflects knowledge about formerly performed IFC inspection of possible unauthorized internal information flows. Knowing a *trust level* of each service, sensitive information can be disseminated to trusted applications (services) only. The main problem of this approach is no guarantee that a trust level (arbitrary assigned to the service by any external entity) is appropriate, since no global trust certification authorities exist so far. Moreover, the SOA paradigm allows the implementation of a given service to change without any notice to the service consumer. This will require to reassign the trust level on each change. It will still be necessary to detect possible implementation changes before using services to forbid information flow that would become unauthorized due to such changes.

All the approaches described above impose strong requirements of internal control of service implementation. Since one of fundamental properties of the SOA model is autonomy of services, provided by third party vendors and being unlikely under any internal control, these approaches are of limited use in service-oriented environments and, in practice, have never been successfully applied there. In this paper, we present a novel approach of incorporating the information flow control in SOA systems. We introduce the concept of Lightweight IFC as a simplification of the general IFC problem, aimed at controlling service-oriented interactions without sacrificing the autonomy of services. Compared to existing solutions, this proposal offers full compliance with the SOA processing model and does not require any source code analysis. We also describe, how information flow control can be managed in an SOA system by using a security policy language and environment. Finally, we describe two possible implementations of the proposed IFC concept using cryptographic keys and poly-instantiated web services.

Our concept of IFC in SOA is closer to *obligation* and *capability* concepts for controlling communication between services, rather than to pure access control. We focus on labeling messages and services with a security level and information category [1]. The security level represents the degree of desired data confidentiality, while the information category determines the possible scope of its usage. Labeling is governed by a new type of security policy rules. Only services with security labels corresponding to message labels are allowed to process messages.

This paper is organized as follows. In section 2, we present the main idea of our proposal. The proposal is then formalized in section 3. In section 4, we describe two possible implementations of our proposal. Concluding remarks are finally given in section 5.

2 Problem Formulation

In SOA, we can perceive the IFC as a problem of controlling how the low and high level data is disseminated among services. If we consider the whole SOA system as a single program and services as processing units, the proposed perspective can be used to project the current IFC approaches to the service-oriented environment. This conforms to the SOA model, since in an SOA environment we are expected to invoke a given service, knowing only its external interface and the presumed functionality it claims to offer. The internal structure of the service is unknown.

Thus, to enforce flow control in an SOA environment we introduce the following IFC properties: (1) proper *isolation* between information belonging to different security levels in communication between particular services, (2) dynamic control of information input (from communication) to services — what we call *invocation IFC*, and (3) ability to prevent a service which has received a sensitive information from passing it outside (e.g. save it to an external storage or send it to other services) without our control (i.e. with different security levels) — *output IFC*.

Next, we will describe how these properties can be effectively provided in a service-oriented environment.

2.1 Lightweight IFC for SOA

The main idea of our proposal is to impose control over the interactions between services with the use of a system-wide IFC policy. Services will be unable to compromise security levels assigned to data, i.e. they will not reuse (resend) the sensitive data with lower security levels. To accomplish that idea we will use a distributed policy enforcement model of ORCA framework [2]. That model (actually basing on former proposals of [6] and [14]) distinguishes several key components which are used to enforce security policy restrictions. *Policy Enforcement Point* (PEP) components are communication intermediaries between services. PEP activities are managed by security policy rules which define restrictions on service accessibility and communication protection. The policy rules are specified in a security policy language. Among other security policy languages, ORCA policy language is the first to support SOA-specific requirements, such as automatic negotiation of security attributes (namely *obligations* and *capabilities*) for dynamic service composition. ORCA policy language can be easily extended for the need of controlling information flow between web services.

In order to fulfill the *isolation* property we extend message format exchanged between services with security *labels* which are composed of a numerical *security level* and an information *category* name. *The Policy Decision Point* (PDP) grants similar labels (*clearance*) to particular services, according to the IFC policy. The *invocation IFC* is enforced by PEP modules, by allowing a service to process a message only if the message label and the service label (or any of the granted labels) match.

The *output IFC* property requires that a services cannot redistribute the received sensitive information in an unauthorized manner. Due to the fundamental assumption of autonomy of services, there is nothing we can truly do in SOA to absolutely control what the service actually does with received information. However, at the communication layer, we can externally control of which part of the rest of the whole system a given service communicates with. In practice there is no need to analyze the internal structure (source code) of services, but only to control all the input and output communication for a given service. Briefly, the *output IFC* requires an output from a service to go only through controlled communication channels.

Having achieved all three IFC properties one can create a SOA system with control of information dissemination, not absolute, yet sufficient for most service-oriented processing. We call this *lightweight* information flow control for web services.

It can be noticed that the concept of Lightweight IFC is a simplification of the general mandatory access control model [12], intended here to control only service-oriented interactions. No "read up" (excess of authorization) and no "write down" (decrease of information security level) are allowed for accessing the information from service invocations and responses during a whole communication session.

A simple illustration of providing IFC in the lightweight manner for a service-based system is presented in Figure 1. This figure depicts two communication

sessions, one initiated by Client1 and the other – by Client2. Each session involves several services. Not all services here are authorized to participate in the data flow from both clients. The information from a particular client request is only accessible to such services that have been granted according clearance labels by the security policy. For example, Service3 cannot process requests from Client1, due to incompatibility of the security label of invocation requests originated from Client1 and Service3's clearance labels. However, Service4 being granted an appropriate clearance level can be involved in that session. Service1 and Service2 are allowed to process information from both sessions.

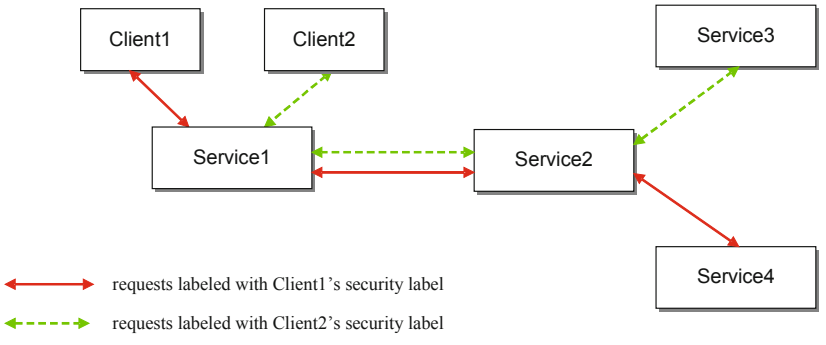


Fig. 1. Lightweight IFC example

More details on how to control the distribution of information from particular client sessions and how to accomplish the isolation of data flows labeled with different security labels are presented in the following section.

3 Formal Model

In order to formalize the proposal we outline here the essential part of the Lightweight IFC model. First, we will specify general rules which organize security level relations. Then, we will present how these rules are implemented in ORCA.

Let us define a security label $\Lambda = \langle \mathrm{L}, \mathbb{C} \rangle$, where L is a security level, and $\mathbb{C}$ is a set of information categories. Information category is not related to any security level. Information categories are necessary to organize information in information domains (such as financial or medical data domains). Security levels are integer values (higher value denotes higher level) organized in relations presented below along with Figure 2. For simplicity of the presentation we consider only a single information category in this figure.

Each interaction message (invocation request or response) is labeled with a security label. Currently, in ORCA, we assign one label to a whole message, however, this can be easily extended to labeling separately particular parts of

the same message. For instance, the security level of the invocation request sent from Service1 to Service2 is denoted by Q_1. The response sent back to Service1 is labeled with a security level denoted by R_1. Initially, for each session, the initiator (actually PEP of the original client starting the session) labels its requests (Q_0) according to the IFC policy ($Q_0 = E_0^{OUT}$, where $E_0^{OUT}$is the security level assigned to the client by the policy).

Each service is assigned an interval of security levels one for incoming messages (for Service1 denoted by $\langle E_1^{INmin}, E_1^{INmax}\rangle$) and one for outgoing messages (denoted by $\langle E_1^{OUTmin}, E_1^{OUTmax}\rangle$).

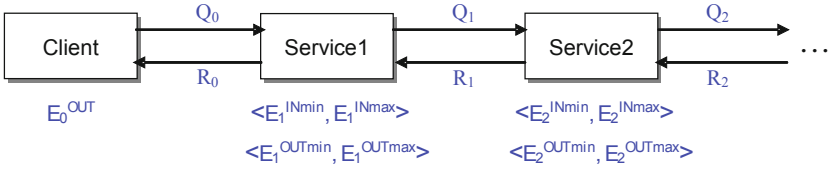


Fig. 2. Security levels assigned to the services

In the Lightweight IFC framework the following relations must always hold:
IFC1: $E_i^{INmin} \leq Q_{i-1} \leq E_i^{INmax}$
IFC2: $E_i^{OUTmin} \leq R_{i-1} \leq E_i^{OUTmax}$
IFC3: $E_i^{OUTmin} \leq Q_i \leq E_i^{OUTmax}$
IFC4: $E_i^{INmin} \leq R_i \leq E_i^{INmax}$
IFC5: $Q_i \geq Q_{i-1}$, i.e. no "write-down" is allowed on nested invocations
IFC6: $R_{i-1} \geq Q_{i-1}$, – no "write-down" on response

For each incoming request, its security level Q_{i-1} is verified against relation IFC1, for outgoing response R_{i-1} – against relation IFC2, for outgoing nested request Q_i – against relation IFC3, and for incoming nested response R_i – against relation IFC4. Additionally, relations IFC5 and IFC6 allow us to hold proper security levels during nested invocations, which is crucial to ensure IFC at the communication layer. Since $Q_{i-1} \leq E_i^{INmax}$ (from IFC1), then there is no read-up on inbound message.

ORCA IFC security policy rules define what security level and information category is a service authorized for. A service can be authorized for a range of security levels and a set of information categories. Sample security policy rules governing IFC for Service1 are presented below in ORCA language:

```
Service1 can accept inbound IFC with category={A}
and {security_level_in_min=2, security_level_in_max=3}.

Service1 must use outbound IFC with category={A}
and {security_level_out_min=3, security_level_out_max=3}.
```

According to these rules the service is not allowed to receive incoming messages other than labeled with information category A and security level 2 or 3. Also,

any outgoing messages will be automatically labeled with that category and security level 3.

4 Implementations

Now, we will complement our proposal with a description of two implementations. The first one uses cryptography to control the information flow between system entities, ensuring that an unauthorized service will not be able to decrypt the received information (message). The second one makes use of distinct instances of each service (one for each authorized level) and redirects the information to the appropriate instance, according to message labels.

4.1 Cryptographic/Cryptographically-Driven Isolation

The first IFC implementation uses asymmetric encryption to restrict access to the information inside messages. The whole body of the SOAP message is encrypted. The idea is to ensure that only services authorized for specific labels are able to decrypt information from given messages.

PEP components, as we already know, are intermediaries which intercept messages and put some security controls on them. These components are now responsible for intercepting and cryptographically secure outgoing messages regarding security label. Each security label has a pair of keys assigned. One key is for encrypting (K_s^{OUT}) and the corresponding one is for decrypting (K_s^{IN}). These key pairs are dynamically generated and distributed by the policy execution environment (PDP in ORCA) to the PEP components during the service registration process. The key distribution is governed by the ORCA IFC security policy rules.

The main advantage of this solution is the ability to process messages carrying data with different security labels by the same service instance, as long as the instance has been granted (by the policy execution framework) appropriate cryptographic keys. However, if multiple communication sessions are processed simultaneously, there is necessity to maintain a session identifier for nested invocations to correctly label the outbound messages (the session identification may involve the service cooperation). Not knowing the source code, we need to trust the service not to tamper the session identifier on nested invocations. In the next subsection, we present another solution which overcomes this drawback.

4.2 Poly-instantiated Services

Another implementation of the proposed IFC concept uses poly-instantiated services. Here, a given service may be launched into multiple instances. Each instance is automatically labeled with a given security label which is designated by the security policy written in ORCA. Each service instance can receive and process only incoming requests with corresponding security labels. The *invocation*

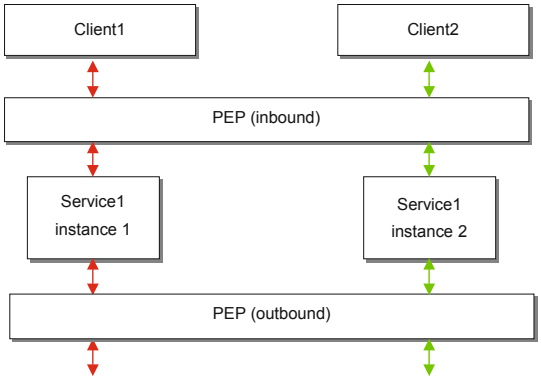


Fig. 3. Multiple service instances

IFC property is achieved by using PEP as an intermediary. Since PEP components control all inbound and outbound messages from and to a given service instance, even untrustworthy services cannot cause information leakage disseminating messages to services which do not have necessary security clearance. The *output IFC* property is achieved by labeling (again via PEP) all the messages sent by a given service instance (i.e. responses or nested invocation requests). It is easy to observe that those outbound messages can further be received only by instances having a proper security label. Finally, the *isolation* property is achieved here, as PEP will not allow a given instance to receive inappropriately labeled requests.

Figure 3 presents a model of poly-instantiated service, where a PEP component controls simultaneous interactions, differently labeled, using two distinct instances of the same service. Each service instance is created and further controlled by PEP. PEP also dispatches incoming requests to the proper service instance with respect to security labels assigned to the incoming requests and each service instance.

The poly-instantiated approach allows us not to worry about isolation of concurrent sessions processed by the same service. Since another instance can be created for each communication session, no additional mechanisms are needed to maintain a proper security level for each session.

The main drawback of this solution, comparing to the previous one, is much larger resource utilization, as several instances of each service obviously consume more resources. There are, however, possible optimizations on the number of service instances. For example, it is possible to use resource utilization metrics and control the number of instances which can be launched and running at the same time.

Currently, a prototype implementation of Lightweight IFC extension of ORCA framework [2] has been developed for WCF platform [9]. We make use of standard `IEndpointBehavior` interface to control creation of service instances. Figure 4 shows the essential part of the implemented application dispatching mechanism. In the presented solution, ORCA PEP gains control over the creation

of service instances with the use of `PerCallInstanceContextProvider` class (line 3) which implements `InstanceContextProvider` interface. The newly created object will dispatch one service instance per each invocation (per-call behavior). Employing our own implementation of `InstanceContextProvider` interface enforces inevitable interception of incoming and outgoing messages (`ServiceMessageInspector`) and allows processing only the messages conforming to the instance's security label (`ServiceParameterInspector`).

Listing 1.1. Control of service instances in ORCA PEP

```
1  public void ApplyDispatchBehavior(ServiceEndpoint endpoint,
     EndpointDispatcher endpointDispatcher)
   {
2     _cacheDispatcher.RunAs = PEPState.Service;
3     endpointDispatcher.DispatchRuntime.
    InstanceContextProvider = new
    PerCallInstanceContextProvider();
    ...
4     endpointDispatcher.DispatchRuntime.MessageInspectors.Add(
new ServiceMessageInspector());
5     foreach (DispatchOperation dispatchOperation in
    endpointDispatcher.DispatchRuntime.Operations)
    {
6       dispatchOperation.ParameterInspectors.Add(
new ServiceParameterInspector());
    }
   }
```

5 Conclusions

In this paper, we have proposed a novel security policy definition framework for the SOA-compliant environments. The presented framework differs from most existing policy frameworks in focusing on aspects crucial for the SOA-based systems, such as full autonomy of service implementation, dynamicity of service compositions and support for interaction obligations and capabilities, among others.

The proposed Lightweight IFC is a solution focused only on controlling which services participate in information processing. Even untrustworthy services can be involved in the processing without the need of source code analysis, impractical or impossible in service-oriented environments. This solution has been experimentally verified using WCF platform [9].

Further work will be focused on development of automated mechanisms for IFC policy verification, concerning analysis of access control and security clearance assignments across the system, and detection of clearance conflicts (incoherent assignments) between particular services.

Acknowledgment. The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

References

1. Bell, D.E., LaPadula, L.: Secure computer systems. Tech. Rep. ESR-TR-73-278, Mitre Corporation (November 1973)
2. Brodecki, B., Sasak, P., Szychowiak, M.: Security Policy Definition Framework for SOA-Based Systems. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 589–596. Springer, Heidelberg (2009)
3. ContentGuard: extensible rights markup language, XrML (2002), http://www.xrml.org/reference.asp
4. Denning, D.E.: A lattice model of secure information flow. Commun. ACM 19, 236–243 (1976)
5. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, p. 11 (1982)
6. ISO/IEC: Information technology - open systems interconnection - security frameworks for open systems: Access control framework (1966)
7. Li, B.: Analyzing information-flow in java program based on slicing technique. SIGSOFT Softw. Eng. Notes 27, 98–103 (2002)
8. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P., Metz, R.: Reference model for Service Oriented Architecture. OASIS Committee Draft 1.0, OASIS Open (2006)
9. Microsoft: Windows communication foundation, http://msdn.microsoft.com/en-us/netframework/aa663324.aspx
10. Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Trans. Softw. Eng. Methodol. 9, 410–442 (2000)
11. She, W., Yen, I.L., Thuraisingham, B., Bertino, E.: The SCIFC Model for Information Flow Control in Web Service Composition. In: Proceedings of the 2009 IEEE International Conference on Web Services, ICWS 2009, pp. 1–8. IEEE Computer Society, Washington, DC, USA (2009)
12. Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1998, pp. 355–364. ACM, New York (1998)
13. Volpano, D.M., Smith, G.: A Type-based Approach to Program Security. In: Bidoit, M., Dauchet, M. (eds.) CAAP/FASE/TAPSOFT 1997. LNCS, vol. 1214, pp. 607–621. Springer, Heidelberg (1997)
14. Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., Waldbusser, S.: Terminology for Policy-Based Management. RFC 3198 (Informational) (November 2001), http://www.ietf.org/rfc/rfc3198.txt
15. Xu, B., Qian, J., Zhang, X., Wu, Z., Chen, L.: A brief survey of program slicing. SIGSOFT Softw. Eng. Notes 30, 1–36 (2005)
16. Yildiz, U., Godart, C.: Information Flow Control with Decentralized Service Compositions. In: ICWS 2007, pp. 9–17. IEEE (2007)

Failure Detection in a RESTful Way*

Dariusz Dwornikowski, Anna Kobusińska, and Jacek Kobusiński

Institute of Computing Science
Poznań University of Technology, Poland
{ddwornikowski,akobusinska,jkobusinski}@cs.put.poznan.pl

Abstract. FaDe service is a novel proposal of failure detection service based on the REST paradigm for the SOA environment. Modularity of architecture allows to build generic service that can be further optimized. FaDe service is fully distributed and provide two communication protocols (gossip, Kademlia) that allow FaDe nodes to cooperate efficiently and make FaDe very scalable. Failure monitoring is based on accrual failure detector which provides flexibility in the context of different client expectations considering the speed and accuracy of detections.

Keywords: failure detection, REST paradigm, distributed systems, service oriented architecture.

1 Introduction

Distributed systems, as a computing platform constitute a very promising research and business area due to their availability, economic aspects and scalability. The intense development of Grids, P2P networks, cluster and high-speed network topologies give the possibility to allocate an enormous amount of resources to distributed applications at a reasonably low cost. Their level of parallelism can improve the performance of existing applications and raise the processing power of distributed systems to a new, higher level. Unfortunately, those systems are failure prone and the probability that a failure occurs during computations is higher than in traditional systems. Moreover, failures of underlying communication network have a great impact on system behavior and condition. There are many techniques to prolong the time in which the component will work correctly or even avoid failure by introducing new rigorous technology procedures. These efforts are often very expensive and cannot ensure full reliability of the system. Therefore, to overcome the problem one should construct a fault tolerant mechanism to detect unavoidable failures and make their effects transparent to a user.

Fault tolerance can be achieved by introducing a certain degree of redundancy, either in time or in space. A common approach consists of replicating vulnerable components of the systems. The replication can guarantee, to a large extent,

* The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 618–627, 2012.

continuous availability of resources and continuity of work, but must be complemented with a failure detection mechanism. Chandra and Toueg presented a concept of failure detectors [2], as an abstract mechanism that supports an asynchronous system model. In [3] authors propose a new type of a failure detector, called *accrual failure detector*. It provides a non-negative *accrual* value representing the current suspicion level of the monitored process, which can be further interpreted by the application. A general survey of the above mentioned failure detectors can be found in [6,10,1].

Service-oriented architecture software is an increasingly popular model of software development nowadays [4]. It provides separation of some independent components (services), which can cooperate with each other through well defined interfaces. The target network environment for this type of architecture is a wide area network environment, characterized by high dynamics of changes of parameters describing it, and susceptibility to failures of its individual elements (nodes, links).

The scalable Web service FaDe, which allows a very flexible failure detection of individual nodes, services running on them, or the unavailability of resources provided by these services is presented below. The proposed solution takes into account various aspects, both those concerning the environment in which failure detection will be performed, as well as those relating to client expectations relative to this type of service.

The remaining part of this paper is structured in the following way. In Section 2 the general concept of FaDe service is presented. Section 3 describes the architecture of the FaDe service node. In Section 4 main aspects of the failure detection module are discussed. External interfaces are described in Section 5. In Section 6 related work is presented. Finally, Section 7 brings concluding remarks and summarizes the paper.

2 FaDe General Concept

Mechanisms that increase reliability are usually built into the application. This approach makes it impossible to reuse the same component, it is also contrary to a service-oriented approach. An independent Web service that provides functionality involving failures detection in a distributed environment can be used by different clients without having to re-implement it in every single individual application. Such an approach also allows for more efficient use of available resources like network bandwidth or power computing.

This approach, however, requires the adoption of some additional assumptions about how the failure detection service works. First, such a service must assume the existence of clients with varying preferences for speed and accuracy of detection failure. This means that a simple binary response will not always be adequate to meet their demands. In this context, a concept in which the mechanism for monitoring services will be based on accrual failure detector [3] was adopted. It gives the possibility of transferring responsibility for the interpretation of the result to the client, which, depending on its individual preferences to decide by itself whether the answer returned by the service means a failure or not.

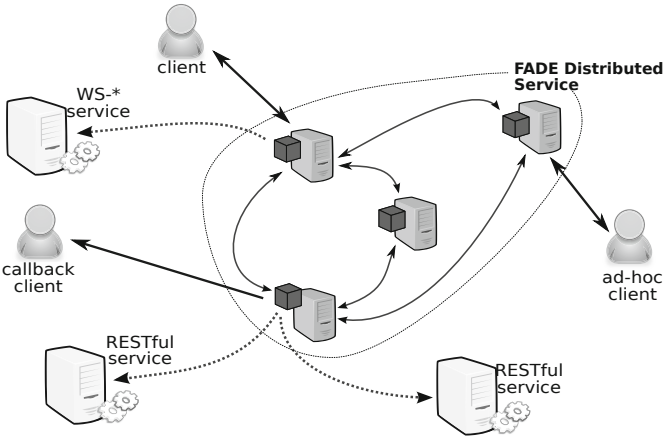


Fig. 1. FADE schema

Another important assumption is the one concerning the ease of setup and adjustment the service to the environment in which it operates. The WAN environmentm characterized by high dynamics and variability, will have an impact on the service efficiency. Therefore, the service itself should be constructed in such a manner that its configuration and tuning to the existing conditions is relatively easy and automatic. Another aspect that can not be overlooked is scalability. In contrast to the services running on the local network, which by definition are limited to a certain area, distributed services dedicated to wide area networks must take into account the possibility of their expansion over a wide geographical area. In this context, scalability is a key element that must be considered.

Finally, one should mention the need of service resilience to its own failures. As stated in the introduction, a distributed environment is prone to various types of failures. If a failure detection service operating in such an environment is to be an element that provides reliability of the application , it should not introduce additional risks associated with its susceptibility to its own failure. Mechanisms that can reduce this risk are e.g. redundancy of key components and decentralization. Responsibility dispersal of monitoring the individual components will minimize the effect of so-called "single point of failure".

Given the above assumptions the failure detection service (FADE), dedicated to the distributed environment was proposed.

Figure 1 presents a sample FADE service schema. In this case, the FADE service consists of four nodes. They are connected and communicate with each other in order to exchange information about connection topology and monitored service states. The FADE nodes monitor three services, both WS-* compliant and REST-based [5]. It is worth noting that FADE nodes can monitor services that provide SOAP or REST type interfaces. It is transparent to the client, which

node monitors a particular service. Moreover one can distinguish three type of clients: standard, ad-hoc and callback which use the FADE service in different manner. In general, clients can inquire the FADE service about the status of the monitored services. Depending on the type of the request, an appropriate action is performed.

3 FADE Node Architecture

A single instance of the FADE service (FADE node) consists of several functional components. Modular architecture allows easy modification of every single component, or even the substitution to a new one that provides compatible interface.

The Internal Communication component is a module that is responsible for communication with other nodes that form a distributed failure detector. Its purpose is to exchange information between the FADE nodes. The information refers to both the state of monitored services and the internal configuration of the FADE service. This communication can be realized with two different protocols: gossip-based and Kademlia-based. As this is out of the scope of this paper, we will not discuss different characteristics of these protocols. From the client point of view, it is transparent which protocol is used.

The Client Interaction component provides external interfaces compliant to WS-* standards and the REST paradigm. The next component,

*The Configuration*manages the service's configuration process. It implements a mechanism that can accept different source of configuration data: XML local file, online Web interface or REST based API calls, and set up the service nodes accordingly.

The Monitoring component is the one directly responsible for monitoring the services. It works by communicating with the monitored services in a manner specified in the configuration with the specified frequency. Based on the results of this message exchange the state of the monitored services can by determined by the core component. The next section describes this mechanism in more details.

4 Failure Monitoring

As mentioned above, the failure detection abstract mechanism is used to detect service failures. To imagine the basic idea behind this mechanism, consider a failure detector FD and services S_1 and S_2. Service S_1 by using FD tries to evaluate if service S_2 is crashed or alive. If failure detector's response is *crashed*, then service S_1 *suspects* S_2. A failure detector that makes no mistakes and eventually suspects all failed services is called a *perfect failure detector*. However, this requirements are very difficult to be fulfilled in practice, due to the asynchronous nature of real systems. So, one can weaken safety (*completeness*) or liveness (*accuracy*) properties to construct an *unreliable failure detector* [2], which can be used to detect failures in an asynchronous system.

A binary value, indicating that the service is working properly or is suspected of the failure is a traditional response of failure detector. It is sufficient when its

clients have similar expectations for speed and accuracy of detection. Unfortunately, when the demands of different clients against these criteria are different, the binary response may be inadequate. As that is the case of the FaDe service, we decided to implement the accrual failure detector [3].

4.1 Failure Detector Response

The typical binary model of a failure detector response has serious limitations when considering failure detection as a generic service. These restrictions come from the fact that some clients require aggressive detection even at the cost of accuracy, while others are interested in more conservative detection with a low level of false suspicions instead of fast but inaccurate decisions. By using binary response generated by a failure detection service, it is generally impossible to fulfill these requirements. It is because the timeout threshold that marks the line between the correct and crashed process should be different.

The accrual failure detector assumes delegation of the decision about the node's crash to the end user application. The failure detection module only provides a non-negative calculated real value, that can be further interpreted by the application. This value represents the current suspicion level of the monitored node (Figure 2). Depending on the threshold assumed by the client, this value can be interpreted both as a correct or a crashed state of the monitored service. The accrual failure detector differs from traditional ones, which interpret the state of the process by itself and return a binary response (correct/crashed) (Figure 2). Thus, in our case, by adopting different thresholds, various clients express their different expectations considering failure detection speed and accuracy.

As an example of an accrual failure detector application the problem of assigning independent tasks to certain nodes can be discussed. The scheduler, i.e. the master node responsible for assigning tasks, must decide which node to choose as a worker — that is the node that executes the task. In case of failure, the scheduler is also responsible for detecting such a situation and reassigning the task to a new worker. At the beginning of task execution, the master node is interested in a fast response (aggressive detection) about the operational status of the worker, while the cost of a wrong suspicions can be neglected. As time goes

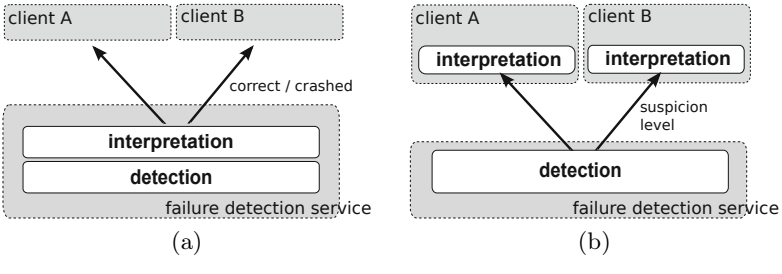


Fig. 2. Interpretation of failure detector response

by, the cost of task abortion and rescheduling due to failure suspicion grows, so false suspicion should be reduced as much as possible. Thus, the master node requires more accurate information about the crashes of workers executing tasks for some time already. In these two situations, the binary response of traditional failure detector is not grained enough, as the master node may require an aggressive detection and a conservative one at the same time.

4.2 Interpretation of the FaDe Response Value

As stated in the previous section, the value returned by the FaDe service must be interpreted by the client. To explain this, lets consider the original function proposed in [3] for the calculation of the suspicion level $\varphi(t_{now}) = -log_{10}(P_{later}(t_{now} - t_{send}))$, where t_{send} is the arrival time of the last heartbeat, t_{now} is the current time, and $P_{later}(t)$ is the probability that a heartbeat will arrive more than t time units after the previous one.

Thus, φ value can be interpreted as follows. Having given threshold $\phi = 1$ and assuming that the process will be considered a suspect if $\varphi \geq \phi$ then the probability of a false suspicion of the process by the detector will be about 10. For $\phi = 2$, the probability will be 1%, for $\phi = 3$, 0.1%, etc.

The FaDe service can use various functions to calculate suspicion level. For the moment, there are five different proposals implemented and extensive test are performed in order to choose the most suitable ones.

4.3 The Monitoring Pattern

A widely used method to perform failure detection is a heartbeat technique. This method is based on periodical probing nodes, in order to determine their status or signaling own status to them. This popular technique is based on the assumption about the maximum delivery time and uses timeouts to decide which node should be suspected as the crashed one. One can distinguish two types of heartbeating depending on who is the active side of the message exchange. The first and the most simple approach is when the node sends messages ("I am alive") about its condition to failure detectors by itself. It is often called the *push* method, as the monitored process continuously advertises (pushes) its state. The second approach is based on interrogation (polling) and is called the *pull* method. In this scenario the failure detector asks ("Are you alive?") about the status of the node and waits for the node's answer. This method is considered to be better than *push* one, because it allows better control of monitoring. The latter approach is also more appropriate in the case of a failure detection service, which should be an active part in the monitoring process. Therefore, the FaDe service performs monitoring using the *pull* model. Since the detection quality, undertaken using this mechanism, is dependent on the characteristics of the network environment in which the monitoring process is being implemented, FaDe service offers a wide range of optional parameters, which allow to adjust this process.

4.4 The Response of the Monitored Service

Despite the fact that the FaDe service is dedicated to monitoring RESTful services, it also allows to monitor services built in accordance with WS-* standards that use the SOAP protocol. However, this was an additional functionality that was realized in order to extend the group of clients interested in using the FaDe and requires the modification of a monitored service. Thus we focus only on services built in accordance with the REST paradigm.

The FaDe service uses HTTP as a communication protocol. In order to minimize the overhead of monitoring we decided to use the HEAD HTTP command in instead of GET or OPTIONS. It should be noted that this is an operation that does not change the status on the server side. Response to the HEAD request should contain only the HTTP header (Fig. 3). It does not contain a resource representation, and thus network traffic is reduced significantly, and the service response with the correct code is sufficient to conclude that the service is available. The OPTIONS method, which returns even less information, was not chosen because of very frequent lack of implementation of this method. The GET method was rejected because of the need to send in response the whole representation of the resource.

FaDe **request:**

```
HEAD / http://test.com/
Host: fd.itsoa.pl
[other headers]
```

Monitored service response:

```
HTTP/1.1 200 OK
[other headers]
```

Fig. 3. Monitoring interaction

By default, the FaDe service can monitor any URI that points to a node, a service or a resource. Optionally, the monitored service may specify a `X-Monitored-Resource` header to inform the FaDe service about an alternative URI, dedicated for monitoring purposes. The choice of HTTP allows the FaDe service to monitor traditional web servers without any modification, because of native support for the HTTP HEAD command response.

4.5 Client Requests

The FaDe service offers three different methods to obtain information about the state of monitored services: *standard query*, *callback query* and *ad-hoc query*. The client of the service chooses one of them according to its own expectations and preferences. These request types complement each other, creating a complete and consistent interface. Below each one of them has been presented and characterized.

Standard query. A basic method for obtaining information from the FaDe service. In this scenario it is assumed that a service client orders monitoring

of certain service. From now on, the service is continuously monitored by FADE. At any time, any client may request information about the state of this service by sending standard query to any FADE node. The node will respond accordingly, or redirect the client request to another FADE node.

Callback query. Sending continuous requests about the status of monitored service to FADE is not always the best solution. Often, the client requires only information about specific change of the service state. This event-based approach allows to react to such a change in the proper way and not engage itself too much in the process of monitoring. Such a scenario can be realized through callback queries. Client registers a callback request and depicts the type of change it expects from the monitored service. If the situation described by the client takes place, it is informed about that fact by the FADE service.

Ad-hoc query. In some situations, the client is only interested in one-time fast information on a service's state. For this purpose, a mechanism of Ad-hoc queries has been made available. These queries are handled by FADE in a different manner than the other two described earlier. Verification of the service availability is carried out in a simplified manner, consisting of sending a single monitoring message. If the service response will be received in predefined time period it is assumed that the service is available. Otherwise, service failure is suspected.

5 Interfaces

When creating a service, it is very important to define appropriate interfaces, that can ensure the interaction with the client service. As the FADE service is consistent with the REST paradigm it implies that the service interface can be described by a tree of available resources. The tree also clearly sets out what resources are available to the client and how it can access and modify them. The resource tree of the FADE service is presented in Figure 4. The client can access specified resources of FADE using appropriate HTTP command and URI.

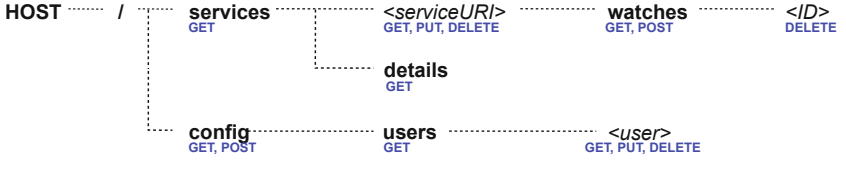


Fig. 4. FADE resource tree

Figure 5 presents a sample interaction scenario. Client sends a request (`GET`) to an appropriate FADE resource (`/services/...`) as it is interested in the state of service located at `http://test.com/`. In response the FADE instance sends the `200 OK` HTTP code indicating, that the information about the requested

Client request:

```
GET /services/{http://test.com/}
Host: fd.itsoa.pl
[other headers]
```

FaDe **response:**

```
HTTP/1.1 200 OK
[other headers]

<service>
   <uri>http://test.com/</uri>
   <correct>0.075322</correct>
</service>
```

Fig. 5. Client query about state of the service

service is available at this node. The body contains an XML structure that describes the details. The tag `<correct>` includes information about suspicion level of monitored service. The closer the value is to 0, the higher the probability the service is working correctly.

6 Related Work

Although FaDe is the first proposal of RESTful failure detection service, there are frameworks for failure detection in the SOA environment that uses SOAP protocol. FT-SOAP [8] and FAWS [7] are the solutions that consist of a group of replica services that are managed by external components. Unfortunately, these frameworks consist of dedicated components that represent single points of failure within the system. FTWeb [11] has components that are responsible for calling concurrently all the replicas of the service and analyzing the responses processed before returning them to the client. Thema [9] is an example of framework that can tolerate Byzantine faults. The Lightweight Fault Tolerance Framework for Web Services [13] achieve fault tolerance through a consensus-based replication algorithm. Finally, WS-FTA [12]is an infrastructure that adapts existing WS-* standards and by providing specialized components allows to tolerate failures.

7 Conclusion

In conclusion, the presented FaDe service allows the detection of failures in a distributed service-oriented environment. Thanks to compliance with the REST architectural style, the service is lightweight and uses available network resources sparingly. Due to resource-oriented approach, its interface is simple, clear and allows easy integration with other services. The ability to accept SOAP requests as well, as the ability to monitor not only RESTful services, makes the FaDe service a very versatile proposal. By using a mechanism based on the accrual failure detector concept, the service provides flexibility in the context of client expectations related to the accuracy and time of failure detection.

References

1. Brzeziński, J., Kobusiński, J.: A survey of software failure detector protocols. Foundations of Computing and Decision Sciences 28(2), 65–81 (2003)
2. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
3. Défago, X., Urbán, P., Hayashibara, N., Katayama, T.: Definition and specification of accrual failure detectors. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN 2005), Yokohama, Japan, pp. 206–215 (2005), http://ddsg.jaist.ac.jp/en/pub/DUH+05b.html
4. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
5. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
6. Freiling, F.C., Guerraoui, R., Kuznetsov, P.: The failure detector abstraction. ACM Comput. Surv. 43(9), 9:1–9:40 (2011), http://doi.acm.org/10.1145/1883612.1883616
7. Jayasinghe, D.: FAWS for SOAP-based web services (2005), http://www.ibm.com/developerworks/webservices/library/ws-faws/
8. Liang, D., Fang, C.L., Chen, C., Lin, F.: Fault tolerant web service. In: Proc. of the 10th Asia-Pacific Software Engineering Conference (APSEC 2003), p. 310. IEEE Computer Society, Los Alamitos (2003)
9. Merideth, M.G., Iyengar, A., Mikalsen, T.A., Tai, S., Rouvellou, I., Narasimhan, P.: Thema: Byzantine-fault-tolerant middleware for web-service applications. In: SRDS, pp. 131–142. IEEE Computer Society (2005), http://dblp.uni-trier.de/db/conf/srds/srds2005.html#MeridethIMTRN05
10. Reynal, M.: A short introduction to failure detectors for asynchronous distributed systems. SIGACT News (ACM Special Interest Group on Automata and Computability Theory) 36(1), 53–70 (2005)
11. Santos, G.T., Lung, L.C., Montez, C.: FTWeb: a fault tolerant infrastructure for web services. In: Proc. Ninth IEEE International EDOC Enterprise Computing Conference, pp. 95–105 (September 2005)
12. Souza, J.L.R., Siqueira, F.: Providing dependability for web services. In: SAC 2008: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 2207–2211. ACM, Fortaleza (2008)
13. Zhao, W.: A lightweight fault tolerance framework for web services. In: Proc. IEEE/WIC/ACM International Conference on Web Intelligence, pp. 542–548 (November 2007)

Compensability of Business Processes

Hubert Gęzikiewicz, Krzysztof Jankiewicz, and Tadeusz Morzy

Institute of Computing Science, Poznan University of Technology, Poland
{Krzysztof.Jankiewicz,Tadeusz.Morzy}@put.poznan.pl,
hubert.gezikiewicz@student.put.poznan.pl

Abstract. When a business process executed in a SOA environment cannot fully achieve its goal, it should perform a compensation of its already completed activities. This method is widely used by the standards related to the executing of business processes in SOA environments. Many of the specifications relative to the types of process coordination, execution languages and notations which are used to design business processes are based on this approach. Unfortunately, there is no specification which provides mechanisms that guarantee the possibility of compensation. The lack of such mechanisms may lead to the situations which in the *BPMN* standard are defined as "hazardous". They occur when an execution of a process can neither be completed nor fully compensated. The result of this process is undetermined, inconsistent with intentions of a designer of a process and can lead to the loss of consistency. These cases often enforce manual engagement in resolving the situation and are a serious problem if we deal with numerous instances of processes.

This article focuses on this issue, presents its analysis as well as a solution to it.

1 Introduction

Most of the standards related to the executing of a long-term business process take into account the mechanisms of compensation. These mechanisms are intended to reverse the effects of the activities performed by the process up to the moment at which the cancellation of this process is decided, and when these activities cannot be simply aborted. Unfortunately, in the environments where business processes are executed concurrently, due to the isolation relaxation, the compensation of the previously completed activities might be impossible. This can lead to the loss of consistency and require manual response. Preventing such cases, or at least reducing them is essential for the business-to-business (*B2B*) or business-to-customer (*B2C*) applications, which operate in a SOA environment.

This article proposes a solution to this problem which takes into account the current industrial specifications and standards related to the coordination of business processes in SOA environments. In section 2 the approaches to compensation presented in standards and specifications have been analyzed. In section 3, the business process model that takes into account these standards has been presented. Based on this model, the problem of providing the possibility of compensation (*compensability*), together with an appropriate motivation example, is

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 628–637, 2012.

characterized in section 4. Section 5 presents the proposal of a solution providing compensability. Finally, the experimental results are presented in section 6. The sections 7 and 8 are respectively discussing related works and a conclusion[1].

2 Compensation and the Standards

The approach to compensation varies between standards. Some of the specifications such as *WS-BA* [9] assume that each Web Service used by the business process is responsible for compensating its activities. In the case of this specification the compensation is triggered by a simple message which is sent to the Web Service. An application that performs the process has no knowledge about how the compensation is performed, and also does not have any impact on it. This approach is similar to the rollback of transaction within the database management system, when the implementation of the rollback is beyond the definition of the transaction.

A completely different solution is used in the business process definition language *WS-BPEL* [4]. In this case, the compensation activities are a part of the process definition. The application is responsible for performing these activities and their implementation.

In the latest version of the specification *BPMN* [1], which is used for business process modeling, another solution has been proposed. On the one hand, a business process designer can define certain compensation for each of the activities undertaken within the process. On the other hand, a coordination type (eg, compliant with the *WS-BA*) can be declared which will be used to coordinate the so-called transactional subprocesses. Therefore it can be stated that the *BPMN* specification tries to combine both approaches with the compensation. Consequently, the system approach in which the knowledge of the compensation is on the side of Web Services and compensation is run through the mechanisms of coordination, and the procedural approach in which the knowledge of the compensation for particular parts of the process is contained in its definition. The inclusion of both approaches in the *BPMN* specification is a strong signal that both solutions are complementary and both should be taken into account. Furthermore, it is the confirmation of the fact that the executing of business processes, which must have certain properties, may not only be based on a simple execution of their definitions, but it also requires (in particular for the transactional subprocesses) the support of the coordination mechanisms.

3 Business Process Model

In further considerations we assume the following model of business process. A business process PB consists of the activities

$$PB = \{A_1, A_2, \ldots, A_n\};\ A_i \in \mathcal{A}$$

where $\mathcal{A}$ is the set of all activities.

[1] The research presented in this article was partially supported by the European Union in the scope of the European Regional Development Fund (ERDF) program no. POIG.01.03.01-00-008/08.

Each activity A_i can have the related compensation activity A'_i. For such case we use the following notation:$A'_i \lhd A_i$. The symbol $\lhd$ indicates that the activity A'_i is a compensation activity related to the A_i activity.

The $\mathcal{A}^c$ denotes the set of activities which have related compensation activities, while the set of the activities that do not have related compensation activities will be denoted as $\mathcal{A}^N$. For such activities the following conditions are met: $\mathcal{A}^N \cup \mathcal{A}^c = \mathcal{A}$ and $\mathcal{A}^N \cap \mathcal{A}^c = \emptyset$.

Moreover, further conditions are also fulfilled:

$$\forall A_i \in \mathcal{A}^c \exists A'_i \in \mathcal{A}^N,\ A'_i \lhd A_i$$

$$\forall A_i \in \mathcal{A}^N \neg\exists A'_i \in \mathcal{A},\ A'_i \lhd A_i$$

Note that the compensation activity may not have its own compensation activities, because it belongs to $\mathcal{A}^N$.

Each activity $A_i \in \mathcal{A}$ may be an action, subprocess or just a collection of other activities.

$$A_i = a_i \vee pp_i \vee \{A_{i1}, A_{i2}, \ldots, A_{in}\};\ A_{ij} \in \mathcal{A}$$

By action a_i we understand the smallest, indivisible, unit of a business process. An example of the action can be a Web Service invocation.

A subprocess consists of activities:

$$pp_i = \{A_{i1}, A_{i2}, \ldots, A_{in}\};\ A_{ij} \in \mathcal{A}$$

We assume that the subprocess can have properties, which are declared and which require the use of coordination mechanisms which are beyond the definitions of subprocess activities. *BPMN* uses the term of a transactional subprocess in this case. We also use this term to a part of the process which uses coordination mechanisms.

The transactional subprocess may result in an approval or cancellation. In the case of a cancellation of transactional subprocess pp_i it is required to perform all compensation activities A'_{ij} such that $A'_{ij} \lhd A_{ij}$; $A_{ij} \in pp_i$.

We assume that the transactional subprocess can have the following properties provided by the coordination mechanisms: **optionality** (which determines the impact of the failure of the transactional subprocess on the success of the parent transactional subprocess), **independence** (which determines the impact of the failure of the parent transactional subprocess on the success of the transactional subprocess) and **compensability**. The purpose of the first two properties is to simplify the definition of business processes, by moving parts of its implementation into the system coordination mechanisms. These properties were discussed in details by us in [10] and are not considered in this paper.

4 Compensability

The third property, which may be required by the subprocess is compensability. As it has been already mentioned, a cancelation of transactional subprocess pp_i

determines the need of compensation for all activities $A_i \in \mathcal{A}^c$, which are the components of transactional subprocess pp_i and that have been completed until the decision to cancel that subprocess has been made. A situation in which only some of these activities will be compensated can lead to inconsistency and therefore is highly undesirable. Compensability aims to eliminate such cases, thus it can be defined as follows.

Definition 1. The **compensability** property of transactional subprocess pp_i is ensuring capabilities to perform all compensation activities A'_{ij} such that $A'_{ij} \lhd A_{ij}$, where $A_{ij} \in pp_i$, until the time when the transactional subprocess pp_i is completed.

It is well known that the primary and most common task of compensation A'_i, such that $A'_i \lhd A_i$, is to reverse partially of fully the effects of activity A_i. Therefore, it can be assumed that the activity A_i *prepares* the possibility of compensation A'_i, and in the case of isolated processing, there are no risks of compensation A'_i failure. Unfortunately, the assumption that the executing of business processes in SOA environments has the isolation property, is neither possible nor desirable [12,11]. Therefore, we have to take into account the concurrent execution of business processes, as well as the isolation relaxation, and we have to assume that the compensation activities may be at risk. Consequently, the following example explaining this problem is going to be considered.

Example 1. Let's assume that the aim of transactional subprocess pp_1 is to exchange a pre-booked hotel room. For this purpose, it has the following two activities: A_{11} – cancellation of a reservation for room X, and A_{12} – making a reservation for room Y. Both activities are performed by invoking the appropriate Web Services. Let's assume that each of these activities has a related compensation activity. For activity A_{11} there is a defined compensation activity A'_{11} which reserves the released room, while for activity A_{12} there is a corresponding compensation activity A'_{12} which cancels the room reservation. The effects of invocations made to each of the Web Services are immediately visible to other processes.

Let's assume that after activity A_{11}, concurrently executed subprocess pp_2 reserves room X, and then commits. Additionally, let's suppose that activity A_{12} could not be completed due to the fact that the room Y was already occupied. As a result, transactional subprocess pp_1 decides to cancel itself. Unfortunately, full cancellation of the process is not possible. Due to the room X being reserved by the process pp_2, the compensation A'_{11} cannot be performed.

The example above shows that ensuring compensability requires additional mechanisms which are beyond the transactional subprocess definition, and which take into account the activities performed by the concurrent processes.

5 Providing Compensability – Subprocess as a Contract

5.1 Contract Negotiations

To provide compensability, the mechanisms responsible for coordination should process the transactional subprocesses as a contract. The parties to this contract

are the application that executes the subprocess and all participants of the subprocess (e.g. Web Service providers) which are responsible for the execution of the compensation activities. As it has been mentioned in previous sections, the information about the compensation activities may come from different sources.

On the one hand, the knowledge about the compensation may belong to the process definition. In this case the compensation is assigned either to a simple activity that can be an action (e.g. single Web Service invocation) or to a complex activity (e.g. a *BPEL* `scope` element), which consists of many sub-activities. On the other hand, this knowledge could be embedded in the implementation of the Web Services used by the process, thus located on the Web Service providers' side. In this case the compensation may be assigned either to a single Web Service invocation or to a series of Web Service invocations (e.g. that are a part of a single *WS-BA* activity). Regardless of the origin of this information, compensation A'_j, can be strictly assigned to the single activity A_i.

The coordination mechanisms responsible for providing compensability must ensure that activity A_i, such that $A_i \in \mathcal{A}^c$, will be performed only if the possibility of the execution of activity A'_i, such that $A'_i \lhd A_i$, is guaranteed. The declarations of such possibility may be granted by the activity providers, in the broad sense, which are responsible for the compensation activities. The granting of such declaration should be the necessary condition to start activity A_i. The lack of such declaration (when it is not possible to guarantee the feasibility of a compensation) must lead either to a suspension of activity A_i (until the moment when the declaration will be granted), or to a cancellation of transactional subprocess pp_i, such that $A_i \in pp_i$. A party which must require such declaration is dependent on the location of the information concerning the compensation. Depending on whether the compensation is declared in the process definition or if it is included in the implementation of the Web Service, the requesting party should be an application which executes that process or the provider of this Web Service.

The feasibility declaration of compensation A'_i will be denoted by $\mathcal{D}(A'_i)$.

According to the standards mentioned earlier, compensation activity A'_i, such that $A'_i \lhd A_i$, can take place only for activities A_i, which have been completed. Therefore, the coordination mechanisms, which are responsible for compensability, should provide the opportunity to exempt the provider of A'_i from the $\mathcal{D}(A'_i)$, when activity A_i is aborted.

Regardless of which party requests $\mathcal{D}(A'_i)$, it is always obtained for the particular instance of transactional subprocess pp_k such that $A_i \in pp_k$. It is very important because these declarations must be valid until the end of the instance of transactional subprocess pp_k. The completion of the transactional subprocess instance is connected with the termination of the contract and therefore, it allows to exempt the provider of A'_i from $\mathcal{D}(A'_i)$.

In summary, the processing of activity $A_i \in \mathcal{A}^c$ should be implemented as follows: **(1)** Before the start of activity A_i, a request for $\mathcal{D}(A'_i)$, where $A'_i \lhd A_i$, is sent to the provider of activity A'_i. **(2)** Upon receipt of $\mathcal{D}(A'_i)$, activity A_i can be launched. **(3)** If the activity A_i is aborted and canceled then the information,

which exempts the provider of A'_i from the given $\mathcal{D}(A'_i)$, is sent to the provider of activity A'_i. **(4)** In the case of aborting or committing of transactional subprocess pp_k, the information about the subprocess completion is sent to each provider of activity A'_i, such that $A_i \in pp_k$.

5.2 Maintaining the Contract

The task of the provider, which has compensation A'_i and operates on the basis of the coordination mechanisms providing compensability of subprocesses, is not only to verify whether the A'_i execution is possible and to grant $\mathcal{D}(A'_i)$, but also to guarantee that possibility for the entire duration of transactional subprocess pp_k, which contains the activity A_i, such that $A'_i \lhd A_i$.

In section 4 has been mentioned that the primary risks to the compensation are the lack of isolation and concurrent processes performing activities threatening the compensation. For these reasons it may be impossible to fulfill declaration $\mathcal{D}$ which was given earlier. This means that the provider of A'_i should have an impact on the execution of all the activities that might prevent the execution of compensation A'_i.

We have to note that the description of how to determine if activity A_j may prevent the execution of compensation A'_i is an issue that goes beyond the scope of this paper, for example it may be related to semantic conflicts.

A service provider who receives a request to perform activity A_j which may prevent the execution of compensation A'_i may respond in several different ways. In the simplest scenario it may suspend activity A_j until the completion of subprocess pp_k which contains activity A_i and which might require the execution of compensation A'_i. However, in case of long-term business processes which use autonomous activities provided by the service providers, other solutions are required. Most authors of works which relate to the control of concurrent business processes [3,11,12] argue that, in SOA environments, the solution should be based on the optimistic approach, and the locks should be significantly reduced or eliminated.

When we use the optimistic approach, we can take into account the fact that activity A_j, which introduces the risk to compensation A'_i may belong to $\mathcal{A}^c$. In this case, the form of compensation activity A'_j may be used by the service provider to choose an appropriate response to the execution request of activity A_j.

In order to consider the issue the example 1 will be analysed again. Let us assume that activity A_{21} of the concurrently executed subprocess pp_2 made reservation of room X. Compensation activity A'_{21} for activity A_{21} cancels the room reservation. The execution of activity A_{21} introduces the risk to compensability for transactional subprocess pp_1, but as long as the subprocess pp_2 is active, there is a possibility to eliminate this risk by the cancellation of subprocess pp_2 and the compensation of its activities.

Considering the above, we can distinguish two types of risks to the compensability: *relative* and *absolute*.

- If activity A_j, which introduces the risk to compensation A'_i, has a compensation A'_j which eliminates this risk, then we talk about a *relative* risk to the compensability.
- If activity A_j, which introduces the risk to compensation A'_i, has a compensation A'_j, which does not eliminate the risk, or does not have any compensation, or has a compensation A'_j which is unknown, then we talk about an *absolute* risk to the compensability.

When the provider receives a request to perform activity $A_j \in pp_k$, which introduces a relative risk to the compensability of subprocess pp_i, activity A_j may be started, however only if the coordination mechanisms allow the control of subprocess pp_k approval. In this case the execution of activity A_j causes the approval of subprocess pp_k to be dependent on the approval of subprocess pp_i. Subprocess pp_i is, in this case, preceding subprocess pp_k. Cancellation of subprocess pp_i forces the prior cancellation of subprocess pp_k, while the approval of subprocess pp_i is a necessary condition for the approval of subprocess pp_k.

In summary, the execution request of activity A_j, belonging to subprocess pp_m, which is sent to the provider of this activity should be handled by the provider in the following way: **(1)** It is verified whether activity A_j introduces a risk to compensation A'_i for any active subprocess pp_n, where $A'_i \in pp_n$, and there are active $\mathcal{D}(A'_i)$. **(2)** If activity A_j does not introduce any risk to any compensation, then the execution of activity A_j can be started. **(3)** If there are active subprocesses pp_n, for which activity A_j introduces a risk to the subprocess compensability, then for each subprocess pp_n it is determined if the risk is relative or absolute. **(4)** If there are absolute risks to the compensability of any subprocess pp_n, then the execution of activity A_j must be suspended until the completion of each subprocess pp_n. **(5)** If there are only relative risks to the compensability of subprocesses pp_n, then the execution of activity A_j can be started. In this way, each subprocess pp_n becomes preceding to subprocess pp_m.

At the approval (or cancellation) stage of the subprocess, this solution requires the cooperation of all the activity providers, which are used by the subprocess, and the application, which is responsible for the execution of the subprocess. The cooperation is required due to the fact that the execution of the activities which introduce relative risks to the compensation couses the subprocess to have other preceding subprocesses, on which it will be dependent. The cooperation may be implemented in accordance to the two-phase commit protocol. Its first phase is used to obtain the confirmation from providers that the subprocess has no other preceding subprocesses and could be committed, while the second phase is intended to send the confirmation to providers about the fact of the subprocess completion.

6 Experimental Results

The coordination mechanisms responsible for the compensability of the transactional subprocesses have been implemented and their impact on processing of business processes in SOA environments has been tested. For this purpose, we

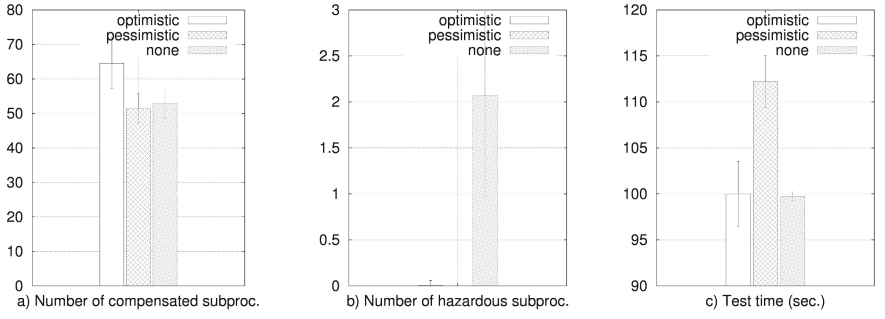


Fig. 1. Test results

have used the *BProCORE* (*Business Process CORdination Environment*) [10], which has been created under the project IT-SOA [5].

Testing scenarios were as follows. Transactional subprocesses were run in 10 parallel series. Each series consisted of 20 subprocesses. The Web Services provided by a single service provider (the hotel) was used by each subprocess. The aim of the process was to replace the booked hotel room. Thus, each subprocess invoked the Web Service two times: at the beginning to release the reservation of a random room, and then to reserve another random room. If both operations were successful then the subprocess was committed, if the second operation failed then the subprocess was aborted by an appropriate compensation. The hotel had only twenty rooms. 10% of them were occupied at the beginning of each test. Due to the concurrent execution of subprocesses there were situations when the compensation activities were at risk. The purpose of the tests was the evaluation of coordination mechanisms responsible for compensability.

Three different configurations of the test environment were applied. In the first one, our coordination mechanisms, based on the optimistic approach, were used. In the second configuration, the pessimistic approach was used. In this configuration any activity, which introduced a risk to the compensation, was blocked. In the third one, the coordination mechanisms were not used.

The test results are shown in figure 1. The first chart (a) shows the number of compensated subprocesses. The second one (b) presents the number of subprocesses, which finished in a hazardous state. The third graph (c) shows the duration of the test. The configurations described above are marked on the figure as: *optimistic*, *pessimistic* and *none*, respectively. The test results lead to the conclusion that the coordination mechanisms can significantly reduce the number of subprocesses, which end in a hazardous state. Unfortunately, the mechanisms of coordination lead to the cases in which the processes are dependent on each other. In such cases, withdrawal of one of the subprocesses leads to the withdrawal of all the preceded subprocesses. This leads naturally to a reduction in the number of processes that are successfully committed. The observed percentage increase in the number of processes that need to be aborted as a result of the coordination mechanisms is 22%, which is about 6% of the total number of the

subprocesses in the test. The tests also included the pessimistic approach. The obtained results show, that it guarantees the compensability of subprocesses and does not lead to cascading aborts. Unfortunately, these advantages come at a price of a very poor performance.

7 Related Works

Recently a number of works related to the coordination mechanisms of concurrent business processes in SOA environments have been published. Some of them consider the general coordination of transactional processes by focusing on the need to simplify their implementation [8] or refer to the atomicity failure of composite Web services [6]. Only a few of them [7] [2] propose concrete solutions which aim at providing the coordination mechanisms that maintain consistency in Web Services Transactions. The authors of these papers propose the solutions that are based on *WS-BA* specification and extend its protocols by additional messages that allow to take transactional dependencies into account. We argue that these solutions are limited to the cases in which compensation depends on the service, which activity has to be compensated. The approach proposed in this paper is more general and closer to the real needs of the developers of business processes. It also takes into account the situations where the definition of compensation is a part of a process definition, as it is in the case of *BPMN* and *BPEL* industrial specifications.

8 Conclusion

The conducted experiments have shown that the mechanisms of the subprocesses coordination designed and implemented by us significantly reduce the number of subprocesses, which end up in a hazardous state. Therefore, it may be useful in SOA environments where compensability is essential.

It should be noted that our work on coordination mechanisms that provide compensability will continue. The coordination mechanisms developed by us, provide compensability but do not guarantee it. The reason for this is the optimistic approach. It allows a situation in which the transactional subprocesses are dependent on each other – a graph of their precedence creates a cycle. In the case of cancellation of one of these subprocesses, there is no correct order of their compensation - one of these subprocesses has to end in a hazardous state. The simplest solution is of course the use of a locking approach, however, due to a low performance and the inappropriateness of this solution for a SOA environment, it is not satisfying.

References

1. Business process model and notation (bpmn) version 2.0 (January 2011)
2. Alrifai, M., Dolog, P., Balke, W.-T., Nejdl, W.: Distributed management of concurrent web service transactions. IEEE T. Services Computing 2(4), 289–302 (2009)

3. Alrifai, M., Dolog, P., Vej, F.B., Dk-Aalborg, Nejdl, W.: Transactions concurrency control in web service environment. In: ECOWS 2006: Proceedings of the European Conference on Web Services, pp. 109–118. IEEE (December 2006)
4. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language (April 2009)
5. Ambroszkiewicz, S., Brzeziński, J., Cellary, W., Grzech, A., Zieliński, K. (eds.): SOA Infrastructure Tools - Concepts and Methods. Poznan University of Economics Press (2010)
6. Bhiri, S., Perrin, O., Godart, C.: Ensuring required failure atomicity of composite web services. In: WWW, pp. 138–147 (2005)
7. Choi, S., Jang, H., Kim, H., Kim, J.-S., Kim, S.M., Song, J., Lee, Y.-J.: Maintaining Consistency Under Isolation Relaxation of Web Services Transactions. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 245–257. Springer, Heidelberg (2005)
8. Erven, H., Hicker, G., Huemer, C., Zaptletal, M.: The web services-businessactivity-initiator (ws-ba-i) protocol: an extension to the web services-businessactivity specification. In: IEEE International Conference on Web Services, ICWS 2007, pp. 216–224 (July 2007)
9. Freund, T., Little, M.: Oasis web services business activity version 1.2 (February 2009)
10. Jankiewicz, K., Morzy, T., Kujawiński, K., Mor, M.: Transaction mechanisms in complex business processes. Control and Cybernetics (2011) (special issue) (in preparation)
11. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. The Coad Series. Prentice Hall, Upper Saddle River (2004)
12. McGovern, J., Sims, O., Jain, A., Little, M.: Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations. Springer-Verlag New York, Inc., Secaucus (2006)

A Developer's View of Application Servers Interoperability

Paweł Lech Kaczmarek and Michał Nowakowski

Gdańsk University of Technology,
Faculty of Electronics, Telecommunication and Informatics,
Naturowicza 11/12 Str., 80-233 Gdańsk, Poland
pawel.kaczmarek@eti.pg.gda.pl, micnowak1@pg.gda.pl
http://www.eti.pg.gda.pl/~pkacz

Abstract. The paper describes analysis of application servers interoperability that considers both the available level of integration and the required level of development complexity. Development complexity ranges from simple GUI operations to changes of undocumented features in configuration files. We verify if an integration can be established on a given level of development complexity, rather than verify if it is objectively feasible. The information indicates whether an integration task can be performed by a non-expert developer, which influences development cost and effort. We focus our work on the Web services standards as leading solutions in service integration. We designed a dedicated test environment that covered Web services based application servers, and helper tools for communication monitoring. We present results and conclusions from performed experimental studies. Detailed results were registered in a web system that supplies descriptions covering specification of application servers, used standards, and configuration options.

Keywords: interoperability, SOA, Web services, heterogeneous systems.

1 Introduction

The Service Oriented Architecture (SOA) approach improved development of applications but requires resolution of interoperability issues, as services are usually deployed on heterogeneous runtime platforms. Web services standards (WS*) [3] were proposed and widely adopted in SOA applications, which significantly improved the integration process.

Despite the general success of WS*, difficulties in effective service integration still exist. Standards contain open points that may be freely interpreted by vendors. A vendor may include specific extensions or implement only selected standards and configuration options [8]. Integration of concrete services is a challenging task that requires detailed analysis of configuration options, software versions and selection of standards.

Considering the difficulties, we performed experimental interoperability verification of WS* based application servers, which covers analysis of achieved level

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 638–647, 2012.

of integration as well as analysis of required level of development complexity. The complexity may range from a simple use of IDE GUI options, through manual modifications of standard configuration options, to hacking undocumented features in runtime platforms. Each test is described in detail by specifying used application servers, WS* standards and detailed configuration options.

The results are useful in improvement of the software engineering process as they describe the complexity of development using concrete standards and application servers. The complexity, in turn, influences the whole development process, including cost, time and necessary skills of the development team.

The rest of the paper is organized as follows. The next section describes related work in Web services interoperability. Sect. 3 discusses interoperability attributes and our selection of standards and application servers. The infrastructure for interoperability testing is described in Sect. 4. Sect. 5 overviews results of experiments. Finally, Sect. 6 concludes the paper.

2 Related Work

Interoperability analysis is a wide research area that supplies results in both academic and industrial fields. [11] overviews existing interoperability metrics and proposes a contemporary interoperability definition as an outcome of approximately thirty definitions proposed during decades of research. This work defines interoperability as the ability of systems, units, or forces, to provide services to, and accept services from, other systems, units, or forces, and to use the services so exchanged to enable them to operate effectively together.

2.1 Interoperability Metrics and Levels

Usually, metrics define levels of interoperability that represent different aspects of system integration; ranging from low-level data transmission to integration of enterprise applications. For example, Levels of Information Systems Interoperability (LISI) [11] defines five interoperability levels: isolated, connected, functional, domain and enterprise. [9] proposes an interoperability assessment model that covers definitions of interoperability levels and evaluation functions.

Ontologies and the semantic description of information are recognized as important mechanisms for automated integration of software systems. The GIS3T concept (Semantics, Standards, Science and Technology) [21] leverages Semantic Web and Web services for data fusion, integration and management of information. [5] proposes a dedicated Glossa language that enables semantics in automated exchanges of information, and integration of the language with SQL-based database engines. [15] compares existing mechanisms of service discovery (WS-Dynamic Discovery, UDDI, ebXML) in the context of service interoperability and semantic description.

[18] discusses both the opportunities, and difficulties, of using semantics in achieving interoperability of enterprice architecture. The work is correlated with

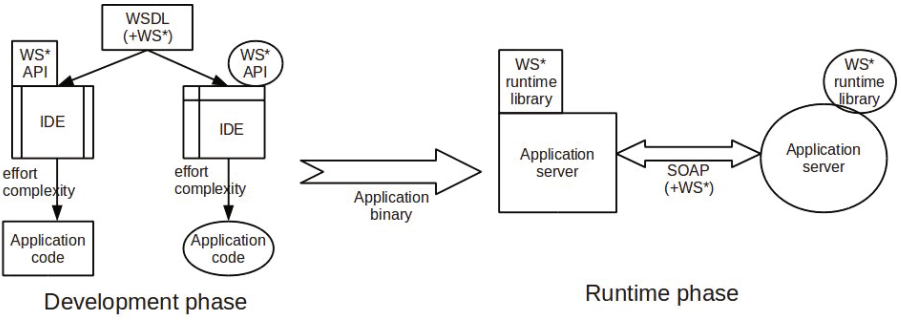


Fig. 1. The general model of the development process and runtime interoperability

European Interoperability Framework [19] that attempts to integrate public administration, enterprises and citizens, on a pan-European level. Another European project, the Interoperability Network of Excellence [20] collected existing results of interoperability analysis from both academic and industrial research.

In our work, we do not propose a new interoperability metric, but rather analyze development and runtime features of WS* integration. Fig. 1 shows the general model of development and runtime phases of integration. The integration considered in this work corresponds to level 1 (Connected) in LISI or level 2 (Data/Object) in LCI (Layers of Coalition Interoperability) [11].

2.2 Development of Heterogeneous WS* Systems

Web services cover over fifty standards, recommendations and notes. They are concerned with both basic communication capabilities (SOAP, WSDL), and extended functionality in the fields of reliability, transactions, security and others [17] [14]. Standards anticipate alternative configuration options, which enables flexible configuration, but may simultaneously result in interface inconsistencies. Considering the proliferation of standards, versions and options, application servers usually implement only a selection of them [2]. Consequently, there is a threat that heterogeneous servers will not interoperate.

Web Services Interoperability Organization has been established by leading vendors of WS* application servers [22] to resolve interoperability difficulties. The organization addresses the problem of incompatibilities in WS* implementations, and issues additional interoperability profiles [6] that refine existing standards and define constraints for communication. [10] [12] are examples of industry-focused analysis that discuss issues of .NET and Java integration.

2.3 Existing Tests of WS* Interoperability

In some cases; standard organizations performed conformance tests of runtime platforms that implement standards, or issued test suites that should verify

standard conformance. The testing process is based on test assertions that specify the contents of messages generated by runtime platforms during WS* communication.

The World Wide Web Consortium executed tests for selected versions of SOAP, WSDL and WS-Addressing standards during development of the standards [7] [13]. SOAP 1.2 and SOAP 2.0 which were equipped with test assertions and candidate implementations and were tested for conformance with the standards. W3C also supplied test suites for the WSDL and WS-Addressing standards. WS-I Organizations supplies Test Assertion Documents that specify the contents of communication for each profile [4]. It is assumed that an implementation must meet the assertions to conform with a profile specification. Microsoft Corporation's issued tests focused on the Windows Communication Foundation environment [16] that aimed at verifying communication possibilities within this specific environment. The Java-based implementation of Metro WS* was also enriched with test assertions for selected issues [1]. We used selected test cases as models for the design of the tests presented in this work.

Existing test suites show some drawbacks in the context of heterogeneous application development. Typically, test results do not specify complexity, workload, and the configuration options required to perform a test. Consequently, non-expert developers may find it difficult to reproduce integrations that require expert knowledge. Our work differs in that we analyze additional features, such as development complexity, and perform cross-platform and cross-standards tests.

3 Interoperability Analysis Model

Most standards are available in different versions and anticipate alternative configuration options. Both are an important issue in service integration, considering that servers may implement standards partially, and option mismatches may occur. Additionally, servers enable you to configure platform specific options that may influence the contents of external communication. Configuration options were set individually for each integration during attempts to invoke a service. Detailed values were registered in test reports available in our web system, as described later.

Considering standards, options and WS* servers, each integration is described by the following data:

- integrated servers - contains specification of server-side and client-side servers, which covers description of runtime platforms, supporting libraries and software versions.
- used standards - specifies which WS* standards were used in an integration
- used configuration options - specifies detailed configuration options that concern either standards, servers or the communication itself, for example used invocation style (RPC/Document) or communication mode (synchronous, asynchronous).

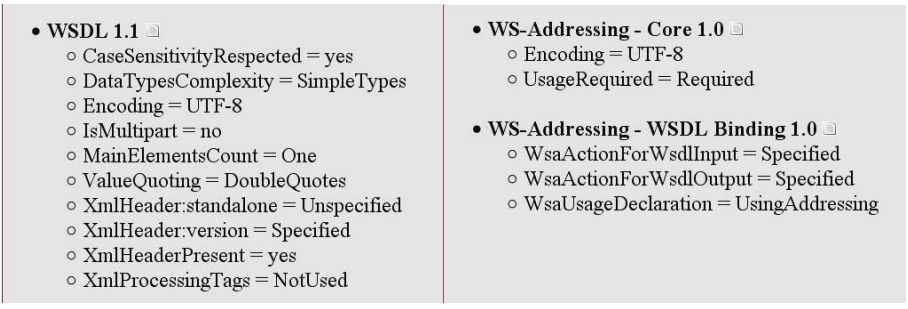


Fig. 2. Exemplary configuration options of WSDL, WS-Addressing Core and WS-Addressing WSDL Binding

A concrete integration is rated for development and runtime interoperability of invocation. Fig. 2 shows exemplary configuration options that may be specified for an integration.

3.1 Selected Servers for Evaluation

The servers that were selected for experiments must implement appropriate WS* extended standards and support a common set of configuration options and versions. Many sources [10] [2] indicate that Java and .NET are leading environments for WS* implementations. Java Web services API includes JAX-RPC and JAX-WS, while JAX-WS is more recent and more interoperable. Analyzing Microsoft .NET products, WCF API is advised, as ASP.NET Web services are considered obsolete and might impair interoperability. Therefore, further considerations were narrowed to JAX-WS and WCF.NET libraries.

Considering the above, we selected the following servers for evaluation:

- JBoss 5.1.0.GA (+ native, Metro, Apache CXF) - supports WS by an auxiliary JBossWS module.
- Apache Geronimo 2.2.0 - uses Apache Axis2 1.5.1 WS* stack implementation, enables hosting on Jetty or Tomcat web containers.
- IBM WebSphere Community Edition 2.1.1.3 - supplies a narrowed implementation of WS* stack as compared to the commercial version, used in basic WS configuration during experiments.
- Microsoft Internet Information Services 5.1 supplies implementation of WS* with custom extensions in the .NET environment, WCF 4.0 stack was used during experiments.

3.2 Interoperability Attributes

The tests aimed at verifying both the achieved level of integration, and the required complexity of the development process. The testing process did not intend to establish integration in each case, but rather to verify if an integration is feasible within a given amount of work and user experience.

We considered two major attributes of interoperability rating in our tests:

Scope describes whether two modules can exchange data correctly. The measure corresponds to achieving a given level of interoperability in more universal interoperability metrics like level 1 (Connected) in LISI (Levels of Information Systems Interoperability) or level 2 (Data/Object) in LCI (Layers of Coalition Interoperability) [11].

Simplicity describes necessary configuration and development effort of achieving an integration. We considered both client-side and server-side configuration in the analysis.

The scope metric uses detailed ratings in range [0..5] associated with relevant runtime events as follows. 5 - no erroneous events were detected during tests, 4 - an internal problem occurred and was handled (logs contain an internally thrown exception, a warning message), 3 - discrepancies from standards in communication (a missing field/attribute, an additional field/attribute, a value different than specified), 1 and 2 - failures in certain conditions (runtime crash, lost communication, hang-up), 0 - failures in all tested conditions.

Simplicity of configuration was calculated from independent ratings of the client-side and the server-side. Each side was rated in range [1..3]. Rate 3 was given if standard configuration and tool-generated code was used. Rate 2 was given if standard API was used for adjustment of code or configuration. Rate 1 was given if the generated code contained errors and low-level API was used to develop communication code by hand. The final rating of client-server integration was scaled to range [0..5], where 5 denotes the simplest configuration and 0 denotes the most difficult configuration.

4 Test Environment

Theoretically, IDE environments should be functional enough to implement adequate integrations. In practice, however, the environments typically cover only selected features of system configuration. For example, they allow us to specify that a given standard must be used, but do not allow us to specify detailed configuration options. Usually, we rely on standard features of IDE environments, but make customized adjustments of IDE-generated code and configuration in some cases. We refrained from using low level API, such as Dispatch in JAX-WS or AXIOM in Axis. The use of this kind of API may improve interoperability, but requires higher development skills, which contradicts with the obvious purpose of minimizing development complexity.

We prepared a WSDL description of services and used adequate tools in different environments (NetBeans, SvcUtil.exe) to generate client-side and server-side communication code. We verified WS-I Basic Profile 1.1 conformance, using the SoapUI tool, for tests with basic SOAP and WSDL standards. Tests that used extended standards (WS-Addressing, WS-ReliableMessaging) were not conformed with WS-I Basic Profile 1.1, as the used profile version did not contain rules for those standards.

Each test case consisted of a simple client and service. Services were the passive side of the tests and were organized according to a common scheme: namespaces

were created for each test service and filled with code skeletons automatically generated from WSDL contracts. Each skeleton was then extended to return appropriate values.

The test environment communicates with the user by a presentation layer, consisting of two web pages. *WSListTestSets*, being the default site, allows us to choose and start execution of one of available test sets. The second page, *WSExecuteTestSet*, presents results after completion of a test. Concrete test cases were inspired by selected tests from [13] [7] [16].

The test environment was installed on a virtual machine in the Virtual Box 3.1.8 runtime platform. The solution provided a number of benefits, including a relatively simple management of environment copies, simple redeployment of basic configuration, and protection against system corruption because of incorrect configuration changes. We ran instances of client-side and server-side using different IP addresses in the same virtual machine system.

The test environment was enriched with sniffing software used for verification and analysis of communication, including NirSoft SocketSniff, TamoSoft CommView, WireShark. Tool selection depended on the particular case of verified servers and standards. It had to be adjusted because of various drawbacks in tool operation; for example, WireShark and NirSoft do not support loopback sniffing in MS Windows, in which case we used TamoSoft and CommView.

5 Aggregated Results of Tests

Using the designed methodology, and the test environment; we performed experiments that covered integration of the selected servers in various configurations. The results of the performed experiments, together with the attempted complexity levels and the achieved interoperability scope have been registered in our system at:

http://www.as-interoperability.eti.pg.gda.pl

During the experiments, we rated the interoperability of each environment from both the client-side and the server-side. We considered the following standards during analysis: SOAP 1.1, WSDL 1.1, WS-Addressing 1.0, WS-Reliable Messaging 1.1, WS-Policy and WS-I BasicProfile conformance. Experiments progressed from basic SOAP communication to more advanced WS* protocols.

5.1 Results for Java-Based Environments

The JBoss Application Server enabled/s us to use different WS* communication libraries. We analyzed three independent libraries during experiments: Oracle Metro, JBoss Native and Apache CXF.

Metro is considered to be a reference implementation of the WS* stack that can/should offer the highest interoperability. Metro works correctly as a client-side library, in nearly all cases reaching a rating of 4 or 5 for both scope and simplicity. It failed in only one case - integration with JBoss Native library. As a server-side library, however, it was able to integrate with approximately

40% of clients. This results from a very restrictive policy if discrepancies from standards are detected in communication received from the client-side. For WS-ReliableMessaging, we were able to integrate the library with only itself. The configurations were verified up to simplicity level 2 or 3.

The JBoss Native library presented lower interoperability in our conducted experiments, as compared to Metro. As a client-side library, it cooperated correctly within WS-I BasicProfile, in which case it reached level 4 or 5 of integration scope. It failed, however, in most other cases, (that is) for standards that extend the scope of WS-I BasicProfile. On the server-side, the library was capable of cooperation with itself and all Metro-compliant services.

Apache CXF was the third library tested within the JBoss Application Server. The library integrated well on the client side within WS-I Basic Profile. In extended standards, the library was interoperable only in some cases. It integrated with Microsoft WCF well, but failed to integrate with Java-based services in most cases. Consequently, integrations received a wide range of rates from 0 to 5. As a server-side library, it integrated/s well as it accepts and handles many discrepancies from standards in client-side communication.

Apache Geronimo is another Java-based implementation tested. The tests covered basic SOAP communication and WS-Addressing only, in which the library presented high interoperability for both client-side and server-side. Development of communication in extended standards requires the use of AXIOM (that is a low-level API). We decided to test integrations within a relatively high development simplicity, and so refrained from using AXIOM. Consequently, integrations in extended standards were rated as scope 0 and simplicity 3. Probably, the integration is possible if advanced programming is applied.

Test of IBM WebSphere AS were performed using Community Edition that supports standards within WS-I Basic Profile only. Therefore, tests were narrowed to basic standards. All integrations were successful for both the client-side and the server-side and were rated as simplicity and scope 4 or 5.

5.2 Results for Microsoft .NET WCF

During experiments, we integrated .NET WCF with various Java-based environments on an assumed development complexity, and effort, level. As a client-side library, .NET WCF integrates seamlessly with Java-based environments for services compliant with WS-I BasicProfile. The integrations were rated as scope 5 and simplicity 4.

We were not able, however, to establish successfull integrations in extended WS* standards in most cases within the attempted development complexity. During communication analysis, we identified an empty wsa:action header sent by WCF, which might be the reason for integration failures. Considering WS-Addressing, JBoss AS + CXF was the only Java-based environment that integrated with WCF. WS-ReliableMessaging communication was successfully established with JBoss AS + Metro in one from many alternative configurations. In other cases, we were not able to establish integration from within the attempted development complexity.

As a server-side library, .NET WCF integrated well with Java-based clients in most cases. It should be noted that the clients were deployed on servers that support relatively new WS* communication standards. .NET WCF requires that clients handle WS-Addressing and WS-Policy apart from the basic SOAP standard, which may cause difficulties in integration with older versions of software. The integrations were rated as scope 5 and simplicity 4.

Asynchronous communication requires an additional comment in the case of .NET and Java WS* integration, as the environments supply different approaches for this communication model: JAX-WS implements the concept internally, using traditional synchronous invocations for communication; the .NET WCF environment supplies a decenter model with alternative notification mechanisms (sampling, delegate callback and implicit callback). Consequently, asynchronous communication between the two environments can not be relied on, in most cases, even if synchronous communication works correctly. Interoperability problems are a known issue, described in[10] a few years ago.

6 Conclusions and Future Work

Results of interoperability tests lead to a number of detailed conclusions. As might be expected, integration of services with basic SOAP/WSDL standards is relatively simple for both Java-based and .NET-based environments. However, establishing integration using advanced WS* standards depends highly on detailed configuration settings and selected standards. Largest difficulties were encountered during integration of .NET and Java-based environments using WS-ReliableMessiging and asynchronous communication. We assumed that services conform to WS-I profiles where applicable. The WS-I organization is an important initiative improving integration of heterogeneous environments.

Experimental evaluation of interoperability in various areas of WS* standards will be an important element of future work. Workflow and transactions related standards seem an interesting area of analysis, as the standards depend on many lower-level ones. The structure of standard dependencies has already been described by academic and industry bodies, which gives background for a systematic classification and testing of higher-level standards. Additionally, we consider categorization of interoperability results to determine configurations in which integration can be achieved with a relatively low effort. We believe that the results will increase the efficiency of software development and advance research in Web services interoperability.

The work was supported in part by the Polish Ministry of Science and Higher Education under research project number N N519 172337.

References

1. Metro Test Harness, https://ws-test-harness.dev.java.net/servlets/ProjectDocumentList
2. Web Services Stack Comparison, Apache, http://wiki.apache.org/ws/StackComparison

3. Austin, D., Barbir, A., Ferris, C., Garg, S.: Web Services Architecture Requirements. Tech. rep., W3C (February 2004)
4. Brittenham, P.: Understanding the WS-I Test Tools. IBM Corporation (2003)
5. Chilukuri, P., Kazic, T.: Semantic interoperability of heterogeneous systems. Tech. rep., University of Missouri, Bioinformatics (2004)
6. Chumbley, R., Durand, J., Hainline, M., Pilz, G.: T., R.: Basic Profile Version 1.2. Web Services Interoperability Consortium, http://www.ws-i.org/Profiles/BasicProfile-1.2-2010-02-16.html
7. Downey, P., Illsley, D., Marsh, J.: Web Services Addressing 1.0 Test Suite. W3C (2003), http://www.w3.org/2002/ws/addr/testsuite/
8. Egyedi, T.M.: Standard-compliant, but incompatible?! Computer Standards & Interfaces (2007)
9. Fang, J., Hu, S., Han, Y.: A service interoperability assessment model for service composition. In: IEEE International Conference on Services Computing, pp. 153–158 (2004)
10. Fisher, M., Lai, R., Sharma, S., Moroney, L.: Java EE and .NET Interoperability: Integration Strategies, Patterns, and Best Practices. FT Prentice Hall (May 2006)
11. Ford, T., Graham, S., Colombi, J., Jacques, D.: A survey on interoperability measurement. In: Proceedings of the 12th International Command and Control Research and Technology Symposium (ICCRTS 2007) (July 2007)
12. Guest, S.: Microsoft .NET and J2EE Interoperability Toolkit. Microsoft Press (2003)
13. Hurley, O., Haas, H., Karmarkar, A., Mischkinsky, J., Thompson, L., Martin, R., Jones, M.: SOAP Version 1.2 Specification Assertions and Test Collection, 2nd edn. W3C (2007), http://www.w3.org/TR/2007/REC-soap12-testcollection-20070427/
14. IBM, Standards and Web services (2010), http://www.ibm.com/developerworks/webservices/standards/
15. Johnsen, F.T., Rustad, M., Hafsoe, T., Eggen, A., Gagnes, T.: Semantic service discovery for interoperability in tactical military networks. The International C2 Journal (2010)
16. Microsoft, Welcome to Web Services Interoperability Plug-Fest!, http://mssoapinterop.org/ilab/
17. Microsoft Corporation, Web Services Specifications Index Page (2010), http://msdn.microsoft.com/en-us/library/ms951274.aspx
18. Niemann, B.: Semantic interoperability community of practice enablement (scope) for enterprise architects. Tech. rep., U.S. Environmental Protection Agency (2005)
19. Novaretti, S.: Eif - european interoperability framework for pan-european egovernment services. Tech. rep., European Commission (2004)
20. Petit, M., Krogstie, J., Sindre, G.: Knowledge map of research in interoperability in the INTEROP NoE. University of Namur (2004)
21. Waters, J., Powers, B.J., Ceruti, M.G.: Global interoperability using semantics, standards, science and technology (gis3t). Computer Standards & Interfaces (2009)
22. Web Services Interoperability Consortium: Interoperability: Ensuring the Success of Web Services (2004)

Traffic Pattern Analysis for Distributed Anomaly Detection

Grzegorz Kolaczek and Krzysztof Juszczyszyn

Institute of Informatics,
Wroclaw University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
{grzegorz.kolaczek,krzysztof.juszczyszyn}@pwr.wroc.pl

Abstract. Network anomalies refer to situations when observed network traffic deviate from normal network behaviour. In this paper, we propose a general framework which assumes the use of many different attack detection methods and show a way to integrate their results. We checked our approach by the use of network topology analysis methods applied to communication graphs. Based on this evaluation, we have proposed a measure called the AttackScore, which assesses the risk of an on-going attack and distinguishes between the effectiveness of the analytic measures used to detect it.

Keywords: Service Oriented Architecture, Security, Anomaly Detection.

1 Introduction

The most intensively explored approach to unknown threats detection is anomaly detection. Anomaly detection can be described as an alarm for strange system behavior. The concept stems from a paper fundamental to the field of security - An Intrusion Detection Model, by Dorothy Denning [4]. The aim of the anomaly detection is discovering of all abnormal states of the system in relation to the network traffic, users activity and system configuration that may indicate violation of security policy [8]. The general idea of protecting computer systems security with anomaly detection mechanisms is very simple, however implementation of such systems has to deal with a lot of practical and theoretical problems. The security assessment of a network system requires applying complex and flexible mechanisms for monitoring values of system attributes, effective computational mechanisms for evaluating the states of system security and the algorithms of machine learning to detect new intrusions pattern scenarios and recognize new symptoms of security system breach [8]. There are three fundamental sets of attributes that are considered in anomaly detection: basic (packet data), content (payload) and traffic (statistics)[5,8].

2 Related Works

The earliest anomaly detection-based approach, proposed by Denning, employs statistics to construct a point of reference for system behavior. The training

R. Wyrzykowski et al. (Eds.): PPAM 2011, Part II, LNCS 7204, pp. 648–657, 2012.

of an anomaly detection sensor is accomplished by observing specific events in the monitoring environment such as system calls, network traffic or application usage, over a designated time period [10]. In many situations one would require constant training of detection system. The example of statistical anomaly detection is e.g. Haystack [11], Intrusion Detection Expert System (IDES)[12] Next-Generation Intrusion Detection Expert System (NIDES) [13]. Research done by Kruegel et al. [14] presents approach to find the description of a system using a payload byte distribution and extracted packet header features. The key aspect of this work is that we provide a quantitative evaluation of different approaches in a single evaluation framework to improve anomaly detection by parallel processing. Some previous work [15] demonstrate that combining multiple detection algorithms does offer an increase in performance over individual detectors. Besides, Gao et al. [3] proposed using an combination of different detection algorithms to build a more accurate model for continuously arriving data and proved theoretical improvement over each single algorithm. However, their work did not consider how to pick the best combination of algorithms. This paper propose distributed traffic pattern analysis to improve the anomaly detection in high speed networks where large quantities of network packets are exchanged by hundreds and thousands of network nodes. The evaluation of detection methods has been performed using a simulation of the Internet Worm attack traces. Compared with earlier works, the presented proposition is more specific in defining the traffic anomaly models.

3 Experiential Evaluation of Distributed Anomaly Detection

Network traffic show some quantitative and topological features that appear to be invariant and characteristic for given network. These distinct features concern topology of network communication, considered as origin-destination flows graph, the distribution of data volumes and the in/out ratio of data sent between nodes [9]. There is also a detectable dependence between worm propagation algorithm, and communication pattern disturbance[8]. Network traffic can be observed and analyzed according to several characteristic values such as: number of bytes send/received per second, number of packets, number of IP destinations, average packet size, etc.. Changing value of these parameters may be viewed as an important source of information about network host of link state. This correlation between network traffic and security breaches has been used in several network intrusion detection systems. For example the following relations between security incident type and observed network traffic parameters change were observed by Anukool Lakhina et al.[1]:

Proposed distributed anomaly detection method will gather information about communication within the network. Then the existing communication patterns will be discovered. The system will be viewed as a graph consisting of nodes and edges which appear if there exists data flow between given pair of nodes. The observation of communication patterns allows to tune the system and track anomalies which are hard to detect on the basis of traffic observations alone.

Table 1. Relations between security incident type and observed network traffic parameters

Security Incident	Traffic anomaly observed
ALPHA	Single source and destination address are dominant with the noticeable number of bytes, number of packets values increase.
DOS,DDOS	Large increase of number of packet with the same (IP,port) pair in the destination address while the distribution of source addresses remains almost unchanged.
SCAN	Increase of flows with the same source address and various combinations of (IP,port) in destination address. Packets with the similar size are dominant.
WORM	Flows with one dominant port in destination address can be observed.

3.1 Modelling Internet Worm for Anomaly Detection Method Evaluation

Internet worms are programs that self-propagate across a network exploiting security or policy flaws in widely-used services [1]. The taxonomies of malware distinguish several types of Internet worms, but there are two major classes of them, scan-based worm and email worms which require some human interaction to propagate and thus propagate relatively slowly. Scan-based worms propagate by generating IP addresses to scan and compromise any vulnerable target computer . This type of worms could propagate much faster than email worms [2]. For example, Slammer in January 2003 infected more than 90% of vulnerable computers in the Internet within just 10 minutes [3]. The basis of our Internet worm modeling is the classical epidemic model [6]. The experimental test bed is assumed to be a homogeneous network any infectious host has the equal probability to infect any susceptible host in the system. Once a host is infected by a disease, it is assumed to remain in the infectious state forever. Two experiments been proposed to evaluate distributed anomaly detection method:

1. Sequential Scanning: This scenario lets each newly infected system choose a random address and, then scans sequentially from there.
2. Hit-list worm: A hit-list worm first scans and infects all vulnerable hosts on the hit-list, then randomly scans the entire network to infect others. It has been assumed that the hit-list comprises the well known address to an infected host.

Common assumptions for all experiments performed are:

- N= 1000 - The total number of host hosts in experimental network
- V=30 - The population of vulnerable hosts in experimental network. It has been assumed that the number of vulnerable hosts is approximately 3% of all population [5]

– I=100 - Average scan rate. The average number of scans an infected host sends out per unit time (time window).

The normal communication activity for experimental network has been modelled using Barabasi scale-free network model [7] with $\gamma = 3$. It was also assumed that the communication is being observed in consecutive time windows with some perturbations during normal network operation which reflect the everyday variance of communication. We have generated realistic communication patterns which join the variance with the properties of a scale-free network. The worm related communication patterns has been added to these normal patterns according to the abovementioned Sequential-scanning and Hit-list scenarios.

3.2 Evaluated Anomaly Detection Algorithms

The idea behind our approach to traffic anomaly detection was to apply structural network analysis in order to compare the topology of communication network during normal operation and during an ongoing attack. We have applied the analysis of role-set structure of a network based on the similarity of link profiles among its nodes. In general structural equivalence measures may be divided into three groups:

– Match measures assuming matching between all pairs of node profiles, usually based on set similarity measures like Jaccard Coefficient etc.
– Correlation measures based on correlation measures applied to node profiles (which are treated as vectors): Cosine, Pearson, Spearman.
– Distance measures measuring the distance between points in n-dimensional space which represent node profiles.

For our experiments we have chosen seven structural equivalence measures:

– Match measures: Jaccard, Phi, Braun and Blanque
– Correlation measures: Pearson, Inner Product
– Distance measures: Euclidean, Bhattacharyya Distance

The interpretation of the results returned by the match and correlation measures is that they are similarity metrics. From the other hand, the distance metrics are the opposite bigger distance stands for more dissimilarity. From this point on, we will refer to all the measures as similarity measures as in fact- we use them to assess how the actual structure of communication network differs from the one emerging from normal system operation. To allow the comparison between the used measures all results were normalized.

3.3 Structural Equivalence Measures during Normal Network Operation

First step in our analysis was to assess the performance of our similarity measures under assumption that there is no attack, and the changes in the communication

Table 2. Structural similarity between communication networks during normal operation

Time step	Jaccard	Phi	Braun, Blanque	Pearson	Inner Product	Euclidean	Bhatt. Distance
1	0,851	0,875	0,992	0,903	0,717	0,770	0,670
2	0,910	0,923	0,998	0,919	0,787	0,802	0,679
3	1,000	1,000	0,990	1,000	1,000	0,864	0,702
4	0,948	0,954	1,000	0,964	0,871	0,813	0,681
5	0,847	0,872	0,987	0,869	0,688	0,782	0,676
Mean value	0,911	0,925	0,998	0,931	0,813	0,806	0,682
Std deviation	0,058	0,048	0,005	0,046	0,113	0,032	0,011

network structure reflect normal operation. The simulation was carried on for six consecutive time windows. For each of these windows the communication network with links reflecting message exchanges between nodes) was created. Table 1 presents the structural similarity between the first and the consecutive time windows as assessed by seven similarity and distance measures.

We assume that the similarity and distance values around the mean value reflect normal network operation. We define an attack as a situation when the similarity differs from the mean value computed on the basis of history more than doubled standard deviation (this attack threshold may be of course tuned in the case of real systems in order to reflect the changes in given network during normal operation). This restrictive assumption may eventually lead to the false attack detection, in the case of data taken from Table 1 this is Bhattacharyya Distance in step 4 or Braun and Braun and Blanque in step 6. In order to avoid false alarms caused any of the measures, we assume that an attack must be confirmed by at least two of them.

3.4 Anomaly Detection

In the second step of our experiments we have checked the influence of Hit-list and Sequential Scanning attacks on the network topology. Fig. 1 presents the results obtained for the first 5 time steps of an ongoing sequential attack.

An immediate consequence of the first infections is the scanning procedure performed by the infected nodes which inevitably leads to the emergence of hubs in communication network which disturbs the network structure and results in visible changes in similarity and distance measures. We can see the growing difference between attacked communication networks and the normal patterns of communication recorded prior to the attack. The only exception is the Inner Product measure, which seems not to distinguish between normal and attacked networks. Note, that distance measures have growing values for older phases of the attack, while match and correlation measures (interpreted as similarity) are decreasing. The same is visible on Fig. 2 which shows similar results for a hit-list

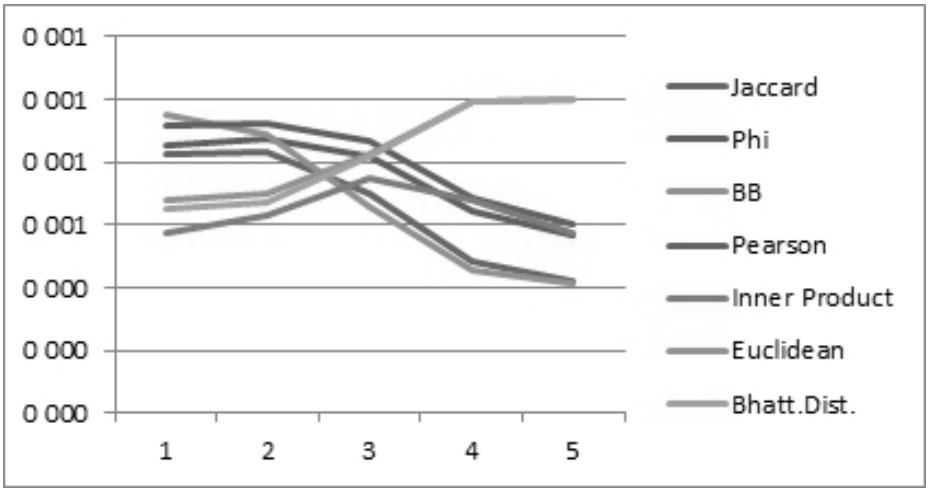


Fig. 1. Similarity and distance measures between normal communication network and the network under sequential attack

Table 3. Attack detection (time steps)

	Jaccard	**Phi**	**Braun, Blanque**	**Pearson**	**Inner Product**	**Euclidean**	**Bhatt. Distance**
Seq. scanning detected in step:	4	4	2	5	6	5	4
Hit-list detected in step:	3	5	2	5	6	5	4

attack. Despite similarities we can also see the differences between Fig. 1 and Fig. 2 they reflect the fact that both modeled attacks have different dynamics, the hit-list worms use the local address lists at first. In result, the number of infected nodes is not so rapid as in the case of hit-list, which is reflected by the moderate (when compared to Fig. 1) change of our measures.

The results presented on the Fig. 1 and Fig.2 may be confronted with the values presented in Table 1, which allows to determine the time step in which each of the measures will report an attack (Table 3).

3.5 Algorithm Aggregation

From Table 3 we can notice that our measures have different performance for each of the considered attacks, there are also differences in the case of attack detection during sequential scanning and hit-list attacks. We define similarity

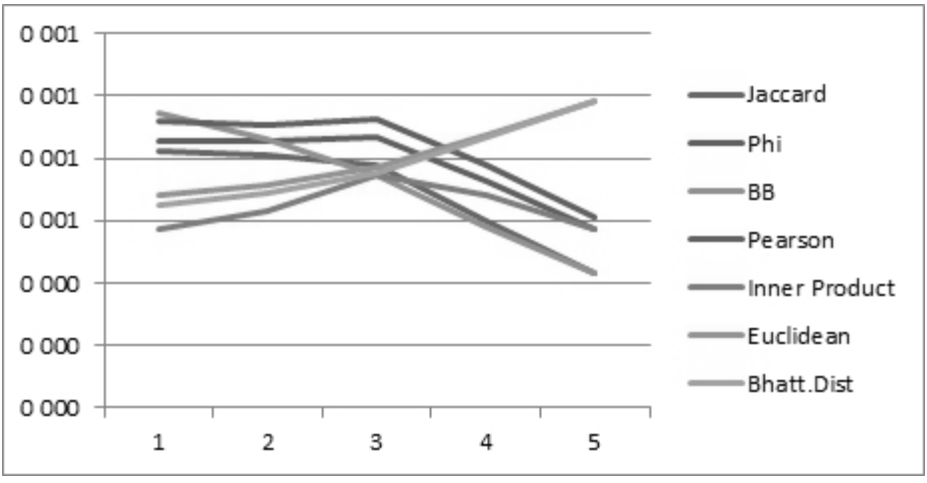


Fig. 2. Similarity and distance measures between normal communication network and the network under hit-list attack

between our measures in terms of their decisions about attack detection. For any two measures m_1 and $m_2$their similarity is defined by comparing the total number of their decisions and the number of decisions in which they have agreed: A_{00} and A_{11} are the number of cases where both measures decided that there is respectively no attack and attack, while A_{01} and A_{10} are the numbers of cases in which they did not agree.

$$SIM(m_1, m_2) = \frac{A_{00} + A_{11}}{A_{00} + A_{11} + A_{01} + A_{10}} \tag{1}$$

Table 4 shows the results for all pairs of measures on the basis of our experiments (the similarity matrix with respect to the decisions of the measures is symmetrical).

Single attack alert (raised by only one measure) may be caused by the normal fluctuations occurring during normal network operation and should not be treated as security breach. We assume that we must get confirmation by at least two of the measures. However, (Tab.4) some of them show close similarity of their results, which fact should be taken into account. In our approach we use a form of "weighted voting" which leads to generation of joint opinion of the measures about the attack. The following rules are applied: 1. At least two measures must positively recognize the attack. 2. When condition 1. is fulfilled a special measure, called Attack Score (AS) is applied to all the measures which raise an alarm.

$$AS = \frac{1}{n_{A^2}} \sum_{j=1}^{n_A} \sum_{i=1}^{n_A} \left(1 - Sim\left(m_i^{attack}, m_j^{attack}\right)\right) \tag{2}$$

Table 4. Similarity between attack detection measures

	Jaccard	**Phi**	**Braun, Blanque**	**Pearson**	**Inner Product**	**Euclidean**	**Bhatt. Distance**
Jaccard	1,000	0,833	0,750	0,750	0,583	0,750	0,917
Phi		1,000	0,583	0,917	0,750	0,917	0,917
Braun, Blanque			1,000	0,500	0,333	0,500	0,667
Pearson				1,000	0,833	1,000	0,833
Inner Product					1,000	0,833	0,667
Euclidean						1,000	0,833
Bhatt. Distance							1,000

Table 5. Attack detection (time steps)

Time window:	**1**	**2**	**3**	**4**	**5**	**6**
Seq. scanning	0	0	0	0,111	0,170	0,217
Hit-list	0	0	0,125	0,148	0,170	0,217

In the above equation n_A is the total number of measures which confirm the attack (lets call them m_1^{attack}, m_2^{attack}, $m_{n_A}^{attack}$). Thus, AS computes the sum of similarities for all possible pairs of attack-reporting measures complemented to 1, then returns their average value. Self-similarity of the measures is zeroed. In this way, if the similarity of the scored measures is high, SA will be significantly lower, then in the case they are behaving in a different way. In result SA promotes the attack reports confirmed by a measures which show different behavior.

The AS reaches its highest value, when all the measures agree about the attack (for our experiment it was 0,217). However it promotes the results returned by the measures which differ from each other in the context of normal network communication. This can be seen in the case of the fourth time window where AS is lower for (Jaccard, Phi, Bhattacharyya Distance) in Seq.Scanning then it is for (Jaccard, Braun-Blanque, Bhattacharyya Distance) in HitList case. This is because the higher difference between the measures recognizing the HitList attack. The higher attack score reflects that it is recognized by the measures which use not the same definition of the structural connection pattern of the network. Moreover, our framework is general and may be applied in the case of measures which differ from each other according to algorithms, nature and the grounding data.

4 Conclusions and Future Work

We have presented an original approach which allows to us different measures for the detection of abnormal network communication patterns. It was tested

on a large network which reflects the scale-free pattern and statistics of the networks detected in different forms of communication . We have also proposed the application of graph structural equivalence measures to the detection of attacks and tested it on the simulated attack occurring in the sample network. Our framework will be further developed in the following directions:

- Detecting the attack type: from the figs 1 and 2 we may notice that different type of attack result in different behavior of our detection. In result the attacks led by various algorithms may be distinguished from each other.
- Instead of simple structural network measures used in our test case, the other sophisticated methods may be applied. Our framework is flexible enough to accommodate and reflect the differences between the measures used.
- SA definition as a weighted voting approach leaves space for checking the interplay between the attack threshold level and the effectiveness of the method.

The first application for our approach will be a SOA system providing educational and administrative services at the Wrocaw University of Technology. The software agents collecting data about normal communication patterns in the system will be developed [15].

Acknowledgements. The research presented in this work has been partially supported by the European Union within the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

References

1. Asokan, N., Niemi, V., Nyberg, K.: Man-in-the-middle in tunnelled authentication protocols. Technical Report 2002/163, IACR ePrint archive (2002)
2. Balasubramaniyan, J.S., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E., Zamboni, D.: An Architecture for Intrusion Detection Using Autonomous Agents. In: Proceedings of the 14th Annual Computer Security Applications Conference (1998)
3. Li, P., Gao, D., Reiter, M.K.: Automatically Adapting a Trained Anomaly Detector to Software Patches. In: Balzarotti, D. (ed.) RAID 2009. LNCS, vol. 5758, pp. 142–160. Springer, Heidelberg (2009)
4. Denning, D.E., Edwards, D.L., Jagannathan, R., Lunt, T.F., Neumann, P.G.: A prototype IDES: A real-time intrusiondetection expert system. Technical report, Computer Science Laboratory, SRI International, Menlo Park (1987)
5. Kolaczek, G., Pieczynska-Kuchtiak, A., Juszczyszyn, K., Grzech, A., Katarzyniak, R.P., Nguyen, N.T.: A Mobile Agent Approach to Intrusion Detection in Network Systems. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds.) KES 2005. LNCS (LNAI), vol. 3682, pp. 514–519. Springer, Heidelberg (2005)
6. Onnela, J.P., Saramaki, J., Szabo, G., Lazer, D., Kaski, K., Kertesz, J., Barabasi, Hyvönen, A.L.: Structure and tie strengths in mobile communication networks. Proceedings of the National Academy of Sciences 18, 7332–7336 (2007)
7. Park, J., Barabási, A.L.: Distribution of node characteristics in complex networks. Proceedings of the National Academy of Sciences of the United States of America 104(46), 17916–17920 (2007)

8. Patcha, A., Park, J.-M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks 51(12), 3448–3470 (2007)
9. Scott, J.: Social Network Analysis: A Handbook, 2nd edn. Sage, London (2000)
10. Anderson, D., Lunt, T.F., Javitz, H., Tamaru, A., Valdes, A.: Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES), Computer Science Laboratory, SRI International, Menlo Park, CA, USA SRI-CSL-95-06 (May 1995)
11. Smaha, S.E.: Haystack: An intrusion detection system. In: Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference, Orlando, FL, pp. 37–44 (1988)
12. Lunt, T.F., Tamaru, A., Gilham, F., Jagannathm, R., Jalali, C., Neumann, P.G., Javitz, H.S., Valdes, A., Garvey, T.D.: A Real-time Intrusion Detection Expert System (IDES), Computer Science Laboratory, SRI International, Menlo Park, CA, USA, Final Technical Report (February 1992)
13. Kruegel, C., Mutz, D., Robertson, W., Valeur, F.: Bayesian event classification for intrusion detection. In: Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV (2003)
14. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A sense of self for unix processes. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, USA, pp. 120–128 (1996)
15. Kołaczek, G.: Multiagent Security Evaluation Framework for Service Oriented Architecture Systems. In: Velásquez, J.D., Ríos, S.A., Howlett, R.J., Jain, L.C. (eds.) KES 2009. LNCS (LNAI), vol. 5711, pp. 30–37. Springer, Heidelberg (2009)

Author Index