Gabrielle Allen Jarosław Nabrzyski
Edward Seidel Geert Dick van Albada
Jack Dongarra Peter M.A. Sloot (Eds.)

LNCS 5544

Computational Science – ICCS 2009

9th International Conference
Baton Rouge, LA, USA, May 2009
Proceedings, Part I

1 Part I

Springer

Gabrielle Allen Jarosław Nabrzyski
Edward Seidel Geert Dick van Albada
Jack Dongarra Peter M.A. Sloot (Eds.)

Computational Science – ICCS 2009

9th International Conference
Baton Rouge, LA, USA, May 25-27, 2009
Proceedings, Part I

Springer

Volume Editors

Gabrielle Allen
Jarosław Nabrzyski
Louisiana State University
Center for Computation & Technology
216 Johnston Hall, Baton Rouge, LA 70803, USA
E-mail: {gallen, naber}@cct.lsu.edu

Edward Seidel
Louisiana State University
Department of Physics and Astronomy
202 Nicholson Hall, Baton Rouge, LA 70803, USA
E-mail: eseidel@lsu.edu

Geert Dick van Albada
Peter M.A. Sloot
University of Amsterdam
Faculty of Science
Section Computational Science
Science Park 107, 1098 XG Amsterdam, The Netherlands
E-mail: {G.D.vanAlbada, P.M.A.Sloot}@uva.nl

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

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F, D.2, D.3, G.1, I.6, I.4, I.3, K.3

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

ISSN 0302-9743

ISBN0 978-3-642-01969-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12687640 06/3180 5 4 3 2 1 0

Preface

"There is something fascinating about science.
One gets such wholesale returns of conjecture
out of such a trifling investment of fact."

Mark Twain, *Life on the Mississippi*

The challenges in succeeding with computational science are numerous and deeply affect all disciplines. NSF's 2006 Blue Ribbon Panel of Simulation-Based Engineering Science (SBES)[1] states 'researchers and educators [agree]: computational and simulation engineering sciences are fundamental to the security and welfare of the United States...We must overcome difficulties inherent in multiscale modeling, the development of next-generation algorithms, and the design...of dynamic data-driven application systems...We must determine better ways to integrate data-intensive computing, visualization, and simulation. Importantly, we must overhaul our educational system to foster the interdisciplinary study...The payoffs for meeting these challenges are profound.' The International Conference on Computational Science 2009 (ICCS 2009) explored how computational sciences are not only advancing the traditional hard science disciplines, but also stretching beyond, with applications in the arts, humanities, media and all aspects of research. This interdisciplinary conference drew academic and industry leaders from a variety of fields, including physics, astronomy, mathematics, music, digital media, biology and engineering. The conference also hosted computer and computational scientists who are designing and building the cyber infrastructure necessary for next-generation computing. Discussions focused on innovative ways to collaborate and how computational science is changing the future of research. ICCS 2009: 'Compute. Discover. Innovate.' was hosted by the Center for Computation and Technology at Louisiana State University in Baton Rouge. Talks and presentations at this conference focused on new applications for high-performance computing, including petascale algorithms, tools and applications, high-speed optical networks such as the Louisiana Optical Network Initiative, or LONI, distributed data management and sharing, and new software programs for biomedical, science and humanities research. The conference included tutorials, a main track session with 5 keynote speakers and 60 accepted, peer-reviewed papers as well as 13 workshops with 138 accepted peer-reviewed papers. Advancing computational science would not be possible without engaging students and young scholars. Through participation in tutorials, workshop and general session paper presentations, the students learned about recent advances and developments in computational science. This year ICCS 2009

[1] Blue Ribbon Panel Report: `www.nsf.gov/pubs/reports/sbes_final_report.pdf`

co-funded with the National Science Foundation a conference student scholarship for around 50 students, mostly from the state of Louisiana. ICCS is committed to helping students and young researchers enhance their professional development through participation in ICCS. During this year's conference two different tutorials were offered to participants: (i) Parallel Performance Evaluation Tools for HPC Systems by Allen D. Malony, University of Oregon, Markus Geimer, FZ Jülich, Andreas Knüpfer, TU Dresden, and Rick Kufrin, NCSA/University of Illinois and (ii) Developing HPC Applications with the Cactus Framework by Erik Schnetter, Frank Loeffler, Eloisa Bentivegna, CCT-LSU. The general main track of ICSS 2009 was organized in about 20 parallel sessions addressing the following topics:

- e-Science Applications and Systems
- Scheduling
- Software Services and Tools
- New Hardware and Its Applications
- Computer Networks
- Simulation of Complex Systems
- Image Processing
- Optimization Techniques
- Numerical Methods

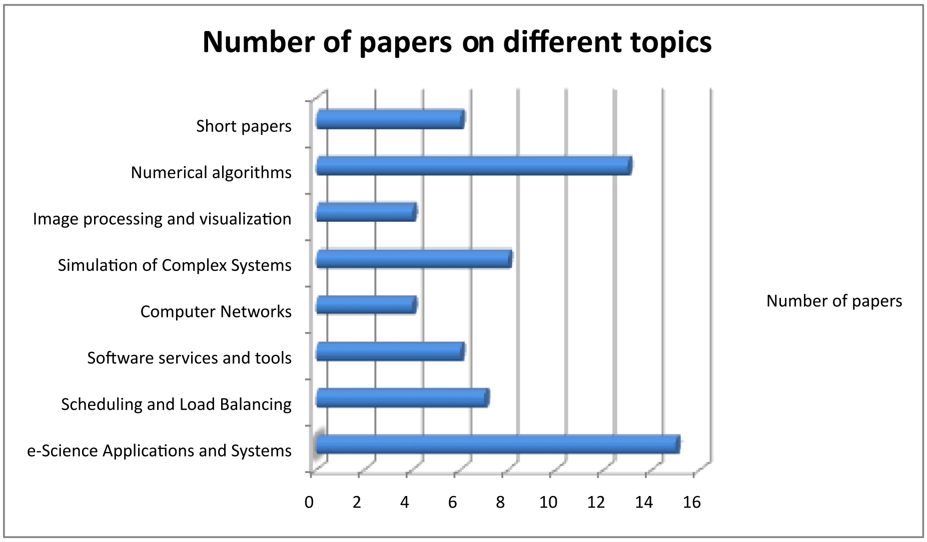


Fig. 1. Number of papers in the general track by topic

Figure 1 presents the number of papers on different topics.

Keynote lectures were delivered by:

- Marian Bubak: *Environments for collaborative applications: An answer to computational science challenges?*

– Janice Coen: *Computational modeling of wildland fire behavior and weather for research and forecasting,*
– Vittoria Colizza: *Computational epidemiology: a new paradigm in the fight against infectious diseases,*
– Peter Coveney: *Grid computing at the petascale,*
– Mark Jarrell: *Massively parallel and multi-scale simulations of strongly correlated electronic systems*

We would like to thank all keynote speakers for their interesting and inspiring talks and for submitting the abstracts and papers for this proceedings volume.

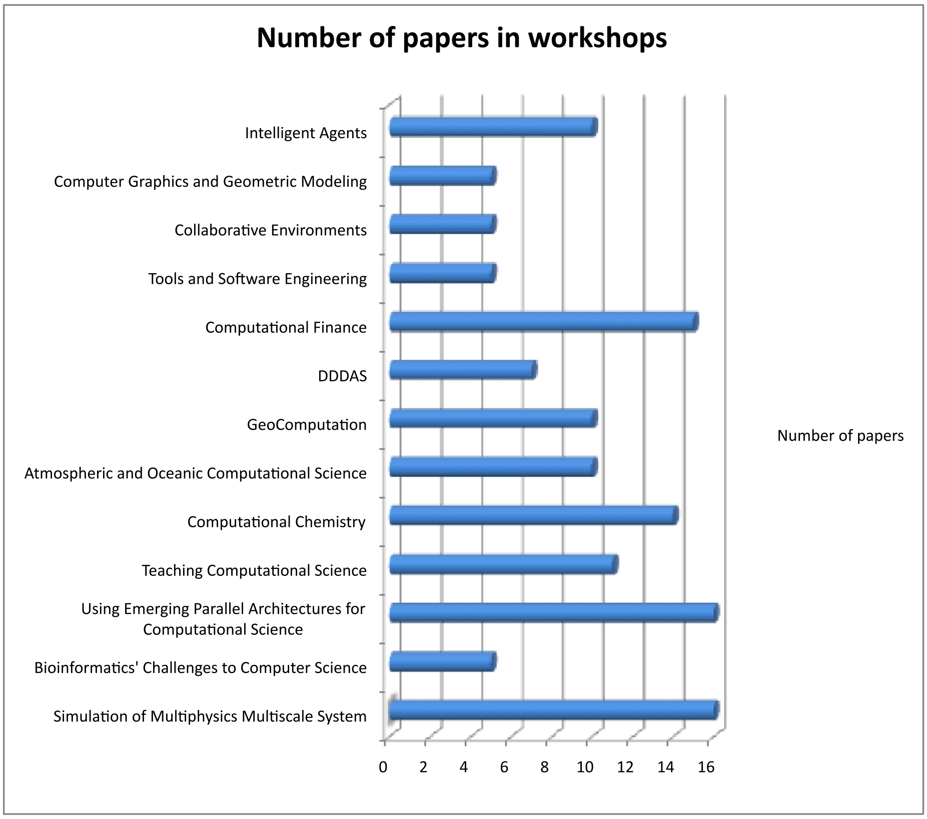


Fig. 2. Number of papers in workshops

The conference also offered 13 workshops:

– Teaching Computational Science
– Computational Chemistry and Its Applications
– Dynamic Data-Driven Application Systems
– Tools for Program Development and Analysis in Computational Science and Software Engineering for Large-Scale Computing

- Simulation of Multiphysics Multiscale Systems
- Workshop on Computational Finance and Business Intelligence
- Bioinformatics' Challenges to Computer Science
- Using Emerging Parallel Architectures for Computational Science
- Collaborative and Cooperative Environments
- Computer Graphics and Geometric Modeling
- Intelligent Agents in Simulation and Evolvable Systems
- Atmospheric and Oceanic Computational Science
- Geo Computation

Figure 2 presents the number of papers in the workshops.

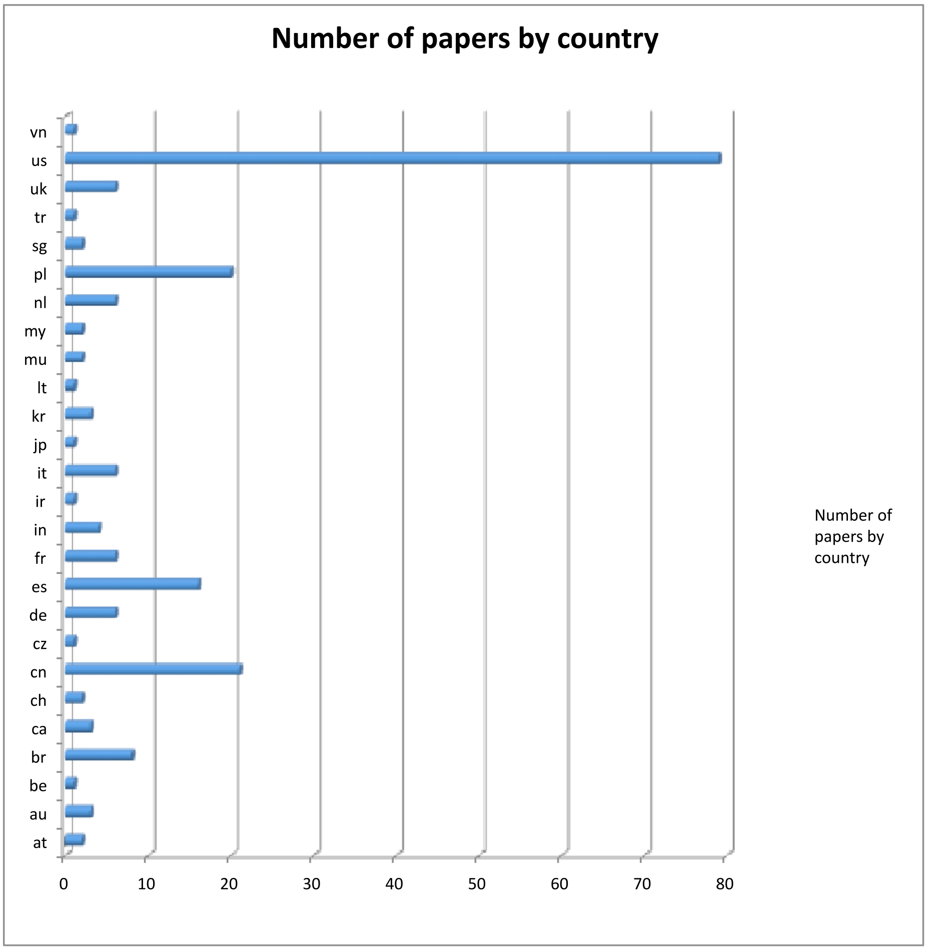


Fig. 3. Number of accepted papers by country

The selection of papers for the conference was possible thanks to the hard work of the Program Committee members and about 390 reviewers; papers submitted to ICCS 2009 received three reviews each. ICCS is a truly international conference, and papers were accepted from 26 countries. The international distribution of papers accepted for the conference is presented in Fig. 3. The ICCS 2009 participants equally represent all continents.

The ICCS 2009 proceedings consist of two volumes; the first one, LNCS 5544, contains the contributions presented in the general track and workshops 5, 7 and 12, while volume LNCS 5545 contains papers accepted for the other workshops. We hope that the ICCS 2009 proceedings will serve as an important intellectual resource for computational and computer science researchers, pushing forward the boundaries of these two fields and enabling better collaboration and exchange of ideas. We would like to thank Springer for a very fruitful collaboration during the preparation of the proceedings.

At the conference the best papers from the general track and workshops were nominated and presented on the ICCS 2009 website; the awards were funded by Elsevier. A number of papers will also be published as special issues of selected journals.

We owe thanks to all workshop organizers and members of the Program Committee for their diligent work, which has ensured the very high quality of the ICCS 2009. We are indebted to all the members of the Local Organizing Committee for their enthusiastic work towards the success of ICCS 2009, and to numerous colleagues from CCT for their help in editing the proceedings and organizing the event. We owe thanks to the ICCS 2009 sponsors: Intel, SiCortex, NSF, Elsevier, CCT and LSU Foundation for their generous support.

We wholeheartedly invite you to once again visit the ICCS 2009 website (http://www.iccs-meeting.org/iccs2009/), to recall the atmosphere of those May days in Louisiana.

May 2009

Gabrielle Allen
Jarek Nabrzyski
Ed Seidel
G. Dick van Albada
Jack J. Dongarra
Peter M.A. Sloot

Organization

ICCS 2009 was organized by the Center for Computation and Technology, Louisiana State University (Baton Rouge, USA), the University of Amsterdam (Amsterdam, The Netherlands) and the University of Tennessee (Knoxville, USA). All the members of the Local Organizing Committee are staff members of the Center for Computation and Technology, Louisiana State University.

Conference Chairs

Conference Chair – Ed Seidel (Center for Computation and Technology, Louisiana State University, USA)
Conference Co-chair – Gabrielle Allen (Center for Computation and Technology, Louisiana State University, USA)
Conference Co-chair – Jarek Nabrzyski (Center for Computation and Technology, Louisiana State University, USA)
Workshop Chair – Dick van Albada (University of Amsterdam, The Netherlands)
Overall Scientific Co-chair – Jack Dongarra (University of Tennessee, USA)
Overall Scientific Chair – Peter Sloot (University of Amsterdam, The Netherlands)

Local Organizing Committee

Gabrielle Allen
Ashlen Boudreaux
Jennifer Claudet
Karen Jones
Jarek Nabrzyski
Susie Poskonka
Kristen Sunde
Debra Waters
Adam Yates
Beata Nabrzyska
Lena Lacińska

Sponsoring Institutions

Intel Corporation
SiCortex
National Science Foundation
Elsevier
CCT
LSU Foundation

Program Committee

J.H. Abawajy	Deakin University, Australia
D. Abramson	Monash University, Australia
G.D. Allen	Louisiana State University, USA
I. Altintas	San Diego Supercomputer Centre, UCSD, USA
M. Antolovich	Charles Stuart University, Australia
E. Araujo	Universidade Federal de Campina Grande, Brazil
E. Bagheri	University of New Brunswick, Canada
B. Baliś	AGH University of Science and Technology, Kraków, Poland
P.K. Baruah	Sri Sathya Sai University, India
A. Benoit	LIP, ENS Lyon, France
I. Bethke	University of Amsterdam, The Netherlands
J. Bi	Tsinghua University, Beijing, China
J.A.R. Blais	University of Calgary, Canada
M. Bubak	AGH University of Science and Technology, Kraków, Poland
K. Bubendorfer	Victoria University of Wellington, New Zealand
J. Buisson	Institut TELECOM, Universite europeenne de Bretagne, France
J. Chen	Swinburne University of Technology, Australia
J. Cui	University of Amsterdam, The Netherlands
J.C. Cunha	Universidade Nova de Lisboa, Portugal
S. Date	Osaka University, Japan
S. Deb	National Institute of Science and Technology, Berhampur, India
B. Depardon	Universit de Lyon - ENS - LIP, France
T. Dhaene	Ghent University, Belgium
I.T. Dimov	University of Reading and IPP, Bulgarian Academy of Sciences, Bulgaria
J. Dongarra	University of Tennessee, USA
F. Donno	CERN, Switzerland
V. Duarte	Universidade Nova de Lisboa, Portugal
J. Ferreira da Silva	Universidade Nova de Lisboa, Portugal
G. Fox	Indiana University, USA
W. Funika	AGH University of Science and Technology, Kraków, Poland
B. Glut	AGH University of Science and Technology, Kraków, Poland
Y. Gorbachev	St. Petersburg State Polytechnical University, Russia

Reviewers

Z. Chen	C. Froidevaux	A. Hoekstra
Z.X. Chen	W. Funika	F.M. Hoffman
X. Chi	S. Furin	J. Huang
B. Chopard	L.M. Gadelha Junior	M. Huebner
Z. Cinar	M. Gallet	E. Hunt
A. Clark	A. Galvez	A. Iglesias
T. Clark	A. Garny	T. Iliescu
E. Constantinescu	T. Gedeon	D.A. Ivanov
E. Conwell	A. Gerbessiotis	H. Iwasaki
C. Cotta	M. Gerrits	R. Jamieson
A. Craik	A. Giesler	M. Jardak
J. Cui	S. Gimelshein	A. Jashki
J.C. Cunha	D. Gimenez	H. Jin
A.C. da Rocha Costa	S. Girtelschmid	D. Johnson
D. Daescu	C. Glasner	J.J. Johnstone
F. Darema	B. Glut	T. Jurczyk
K.K. Das	R. Goh	J. Jurek
S. Date	L. Gorb	R. Kakkar
B.R. De	Y. Gorbachev	A. Kalyanaraman
J.N. de Souza	A.M. Gościński	B.D. Kandhai
S. Deb	A. Goursot	S. Kawata
Y. Demazeau	G.A. Gravvanis	W.P. Kegelmeyer
B. Depardon	C. Grelck	R. Kelly
R. Dew	D.J. Groen	W.A. Kelly
T. Dhaene	P. Gruer	C. Kessler
I.T. Dimov	T. Gubała	C.H. Kim
G. Dobrowolski	C. Guerra	M. Kisiel-Dorohinicki
T. Dokken	X. Guo	J. Kitowski
J. Dongarra	Y. Guo	C.R. Kleijn
F. Donno	P.H. Guzzi	A. Knüpfer
C.C. Douglas	A. Haffegee	R. Kobler
R. Drezewski	F.B.H. Hagelberg.	M. Koda
S. Dua	M. Hamada	I. Kolingerova
V. Duarte	U. Hansmann	J. Kolodziej
W. Dubitzky	M. Hardt	R. Kooima
F. Dufoss	W.W. Hargrove	M. Korek
S. Emrich	J. He	S. Kostov
V. Ervin	T. Heinis	G. Kou
K. Evans	P. Heinzlreiter	J. Koźlak
D. Fedorov	M. Hell	P. Kozyra
J. Ferreira da Silva	D. Henze	M. Krafczyk
T. Ford	V. Hernández	D. Kranzlmüller
G. Fox	P. Herrero	B. Kryza
F. Freitag	L. Hluchý	V.V. Krzhizhanovskaya
H. Freitas	B. Hnatkowska	V. Kumar

A. Lagana
K.K. Lai
R. Landertshamer
D. Lavenier
H. Lee
H.K. Lee
H.S. Lee
L. Lefevre
P. Leong
A. Lewis
A.H. Li
G.Q. Li
J.P. Li
S. Li
H.W. Lim
Z. Lin
L. Liquori
B. Liu
D.S. Liu
J. Liu
R. Liu
W. Liu
Y. Liu
M. Lobosco
R. Loogen
E. Lorenz
P. Ltstedt
M. Low
P. Lu
D. Luebke
W. Luo
C. Lursinsap
T.J. Ma
S. MacLachlan
K. Madduri
N. Maillard
M. Makki
M. Malawski
U. Maran
M. Mascagni
D.L. Maskell
R. Mason
K. Matsuzaki
F. Mavelli
M.L. McCarthy
A.S. McGough
W. Meira
A.S. Memon
Z. Michalewicz
J. Michopoulos
S. Midkiff
R.T. Mills
A. Misra
D.G. Mitmik
G. Moltó
F. Munoz
A.R. Mury
J. Nabrzyski
R.D. Nair
S. Naqvi
A. Nasri
P.O.A. Navaux
E. Nawarecki
Z. Németh
L. Neumann
G. Nikishkov
B. Ó Nualláin
J.T. Oden
A. Oliferenko
D. Olson
M. O'Neill
P. Orantek
G. Ostrouchov
J. Owen
M. Paprzycki
C. Pautasso
B. Payne
T. Peachey
Y. Peng
M.S. Pérez
G. Pfeiffer
D. Plemenos
M. Polak
V. Prasanna
G. Qiu
A. Queiruga Dios
M.R. Radecki
U.R. Radius
B. Raffin
P. Ramasami
A. Rendell
M. Riedel
R. Righi
S. Ringuette
Y. Robert
E.R. Rodrigues
D. Rodríguez García
C. Rodríguez-Leon
F. Rogier
F.-X. Roux
R. Ruiz
M. Rumi
K. Rycerz
A. Sandu
M. Sbert
R. Schaefer
H.F. Schaefer III
J. Schatz
M. Schimmler
B. Schmidt
L. Schnorr
H. Schroder
J. Schroeder
B. Schulze
S. See
M. Segarre
J. Seo
A. Sfarti
H. Shi
Y. Shi
A. Shiflet
F. Silvestri
B. Simo
D. Sinclair
V. Sipkova
P. Slizik
R. Slota
B. Śnieżyński
A. Soldera
A.E. Solomonides
A. Sourin
R. Spiteri
V. Srovnal
A. Stamatakis
V.S. Stankovski

A. St-Cyr
A. Streit
J. Sumitomo
H. Sun
J. Sundnes
H. Suzuki
C. Swanson
C.T. Symons
D. Szczerba
L. Szirmay-Kalos
R. Tadeusiewicz
R. Tagliaferri
W.K. Tai
E. Talbi
T. Tang
X.J. Tang
J. Tao
M. Taufer
M. Taylor
C. Tedeschi
D. Thalmann
L. Tininini
A. Tirado-Ramos
R.F. Tong
A. Tsinakos
P. Tvrdik
B. Ucar
F. Ucun
G.V. Valenzano
A. Valladares
G.D. van Albada
S.J. van Albada
M. van der Hoef
R.R. Vatsavai
J. Villa i Freixa
J. Volkert
G. Voss
H.S. Wahab
D.W. Walker
C.L. Wang
J.J. Wang
M. Wang
Z.X. Wang
R. Weber dos Santos
W. Weijer
R. Wismüller
P.H. Worley
S.Y. Wu
Y. Xue
N. Yan
C.T. Yang
F.-P. Yang
L.T. Yang
J. Yu
I. Zelinka
J. Zhang
J.J. Zhang
L.L. Zhang
B. Zheng
A. Zhmakin
H.A. Zhong
X.F. Zhou
X. Zhu
J. Zola
A. Zomaya
E.V. Zudilova-Seinstra

Workshops Organizers

Third Workshop on Teaching Computational Science (WTCS 2009)

A.B. Shiflet (Wofford College, Spartanburg, SC), A. Tirado Ramos (University of Amsterdam)

Workshop on Computational Chemistry and Its Applications (4th CCA)

P. Ramasami (University of Mauritius), F.H. Schaefer III (University of Georgia)

Dynamic Data-Driven Application Systems - DDDAS 2009

C.C. Douglas (University of Wyoming)

Joint Workshop for Tools for Program Development and Analysis in Computational Science and Software Engineering for Large-Scale Computing

A. Knüpfer (ZIH, TU Dresden, Germany), D. Rodríguez (The University of Alcalá), J. Tao (Forschungszentrum Karlsruhe, Germany)

Simulation of Multiphysics Multiscale Systems, 6th International Workshop

V.V. Krzhizhanovskaya (University of Amsterdam), A.G. Hoekstra (University of Amsterdam)

Workshop on Computational Finance and Business Intelligence

Y. Shi (University of Nebraska at Omaha and Chinese Academy of Sciences), X. Deng (Department of Computer Science, City University of Hong Kong), S. Wang (Academy of Mathematical and System Sciences, Chinese Academy of Sciences)

Bioinformatics' Challenges to Computer Science

M. Cannataro (Magna Græcia of Catanzaro University), J. Sundnes (Simula Research Laboratory, Norway), R. Weber dos Santos (Federal University of Juiz de Fora, Brazil)

Using Emerging Parallel Architectures for Computational Science

B. Schmidt (Nanyang Technological University), D.L. Maskell (Nanyang Technological University)

Collaborative and Cooperative Environments

C. Anthes (GUP, Joh. Kepler University Linz), V. Alexandrov (ACET, University of Reading, UK), D. Kranzlmüller (LMU Munich and LRZ Garching, Germany), J. Volkert (GUP, Joh. Kepler University Linz)

Eighth International Workshop on Computer Graphics and Geometric Modeling, CGGM 2009

A. Iglesias (University of Cantabria)

Intelligent Agents in Simulation and Evolvable Systems

R. Schaefer (AGH University of Science and Technology), K. Cetnarowicz (AGH University of Science and Technology), B. Zheng (South-Central University For Nationalities, Wuhan, China)

Atmospheric and Oceanic Computational Science

A. Sandu (Virginia Tech), A. St-Cyr (National Center for Atmospheric Research), K. Evans (Oak Ridge National Laboratory)

Geo Computation

Y. Xue (London Metropolitan University), F.M. Hoffman (Oak Ridge National Laboratory), D.S. Liu (Remote-Sensing Satellite Ground Station, Chinese Academy of Sciences)

Table of Contents – Part I

Numerical Algorithms

Short Papers

Table of Contents – Part II

e-Science Applications and Systems

Electronic Structure Calculations and Adaptation Scheme in Multi-core Computing Environments

Lakshminarasimhan Seshagiri[1], Masha Sosonkina[1], and Zhao Zhang[2]

[1] Ames Laboratory
Iowa State University
Ames, IA 50011 USA
{sln,masha}@scl.ameslab.gov

[2] Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011 USA
zzhang@iastate.edu

Abstract. Multi-core processing environments have become the norm in the generic computing environment and are being considered for adding an extra dimension to the execution of any application. The T2 Niagara processor is a very unique environment where it consists of eight cores having a capability of running eight threads simultaneously in each of the cores. Applications like General Atomic and Molecular Electronic Structure (GAMESS), used for ab-initio molecular quantum chemistry calculations, can be good indicators of the performance of such machines and would be a guideline for both hardware designers and application programmers. In this paper we try to benchmark the GAMESS performance on a T2 Niagara processor for a couple of molecules. We also show the suitability of using a middleware based adaptation algorithm on GAMESS on such a multi-core environment.

Keywords: Multi-Core, GAMESS, Niagara, Adaptation, NICAN.

1 Introduction

Computational chemistry applications like GAMESS [5] are widely used to perform ab-initio molecular quantum chemistry calculations. These calculations include a wide range of Hartree-Fock (HF) wave function (RHF,ROHF and UHF) calculations. Such calculations are not only complex but also have high computational requirements. These calculations are currently run on SMP clusters where each node consists of single or dual core processors. SMPs can be viewed as a form of NUMA (Non Uniform Memory Access) architecture [4]. NUMA is the design used where each processor in a multi processor environment is provided with a separate memory space and data is being shared between different memory banks. This needs to be handled using separate hardware and software since the data can be distributed over different processor memories and coherency between this data needs to be maintained. The communication cost plays a significant role in this design. Each node in a computing cluster can be considered equivalent to a processor in the NUMA architecture, but with a coupling that is not as tight as that in NUMA. The inter-node latency and bandwidth in clusters is much worse than a normal NUMA machine. Hence when we run an IO intensive application like the conventional GAMESS job, it is very likely to bog down the channel thereby resulting in slower execution times. The problem of remote data access on symmetric multiprocessor (SMP) nodes is avoided by the usage of a native communication layer DDI (Distributed Data Interface) that utilizes the shared memory effectively [13].

A multi-core processor capable of running multiple threads in each core can be used as a single execution environment in itself instead of a SMP cluster. The execution semantics change in such an environment. Each of the threads act as a virtual processor (VP) to the outside world. Thus the user application sees itself running on a multi-processor machine with access to each and every one of them. Each of these VPs include all the architecturally required components to execute a task. These components include registers (both general purpose and special), integer and floating point execution units and can handle interrupts. The execution units are present inside each core of the processor and the VPs belonging to each core share these components. Thus each VP contains a separate instance of the user state. Since the multi-core processor is fabricated on a single chip, the resources such as memory bandwidth, L1 and L2 caches are shared among the VPs. This has a significant impact on the performance of the task being executed.

There have been studies using benchmarking tools to show the performance of SMP clusters which use single core or dual core processors such as the Intel Woodcrest processor [1]. There have also been similar studies on the Niagara processor [1] that extol the advantages of using a multi-core and a multi-threaded processor. An understanding on the performance of an application like GAMESS on a SMP and a multi-core environment like the Niagara processor will be of immense help not only for application programmers but also from processor designers. We compare the performance of GAMESS on these two environments in order to understand the relative advantages and disadvantages of the two. Also, of note is the fact that resource sharing in a multi-core processor running multiple VPs would have a great impact on the performance of a computationally intensive application like GAMESS. Various studies [10], [9], [7] have been done to arrive at the best possible combination of GAMESS processes per node (on a SMP) so as to overcome the resource constraints. It has already been shown in [8] that an adaptation algorithm using a generic middleware tool NICAN [2,6] would improve the performance of GAMESS. We show that this adaptation scheme is of relevance and importance in a multi-core and multi-threaded environment as well.

The rest of the paper is organized as follows. Section 2 describes the workload used and the architecture of the execution environment. Section 3 describes the performance of GAMESS on a SMP cluster and a Niagara processor. Section 4 deals with the adaptation algorithm and the results obtained by using this algorithm on a Niagara processor.

2 Methodology

2.1 Application Workload

General Atomic and Molecular Electronic Structure (GAMESS) performs ab-initio molecular quantum chemistry calculations [5] to perform a wide range of Hartee-Fock (HF) wave function (RHF, ROHF and UHF) calculations. Using Self Consistent Field (SCF) method, GAMESS iteratively approximates solution to the Schrödinger equation that describes the basic structure of atoms and molecules. Numerous GAMESS calculations have parallel implementations utilizing distributed resources like memory and disk storage. The scalability of GAMESS is aided by the use of a native communication layer DDI [13] that takes advantage of shared memory on symmetric multiprocessors (SMP) and reduces the remote data access bottleneck. The SCF method is one of the most computationally intensive parts in the GAMESS execution. It has two implementations, *direct* and *conventional*, which differ from each other in the handling of the two-electron (*2-e*) integrals.

In the *direct* SCF method, the 2-e integrals are recalculated for each iteration and it avoids any I/O bottleneck. In the *conventional* SCF method, the 2-e integrals are calculated once at the beginning of the SCF process and stored in a file on disk for subsequent iterations. These two implementations are interchangeable [8] due to the iterative nature of the process. SCF method also gives a good indication of the processor computation power as well the I/O capabilities of the system on which GAMESS is being run. Thus a GAMESS run on a SMP and a Niagara processor can be favorably compared to get some sort of an opinion on the relative merits of using either of these two architectures. We chose *Luciferin* and *Ergosterol* molecules for testing GAMESS on these two platforms. GAMESS converges in 15 iterations for both *Luciferin* and *Ergosterol.* A *conventional* execution of *Luciferin* requires a storage of almost 3.5GB of files while a *direct* execution consumes 5.65MB of main memory. On the other hand, a *conventional* execution of *Ergosterol* molecule stores 22GB of files and requires nearly 16MB of main memory for the *direct* implementation.

2.2 Architectures Used

We used two different architectures to test GAMESS. One was a SMP cluster of 4 nodes, each node having two dual-core 2.0GHZ Xeon “Woodcrest” CPUs and 8GB of RAM [1]. The nodes were interconnected with both Gigabit Ethernet and DDR Infiniband. Each processor has a shared 4MB L2 cache. It also contains a 32KB L1 instruction and data cache per core.

The second architecture used for testing was the Sun T2 Niagara processor (T2) [11,14]. The T2 processor has a unique architecture which consists of 8 SPARC physical processor cores built in a single chip and each core is capable of running 8 threads. Each of these threads can be considered to be a processor in itself and are called as Virtual Processors (VP). Thus a user application sees itself running on a machine of 64 processors rather than on a processor containing 8 cores. The VPs operate at a frequency of 1167 MHz. Each of these cores contain full hardware support for the eight VPs. There are two integer execution pipelines, one floating point pipeline and one memory pipeline inside a single core that are shared between all the VPs. The eight VPs are divided into two groups of four each with the VPs 0-3 occupying one group and 4-7 occupying the other group. Obviously, the hardware support inside a single core also gets divided accordingly with each group of VP having access to a single integer pipeline and sharing the floating point and memory pipelines. Each SPARC physical core contains a 16 KB, 8-way associative instruction cache (32-byte lines), 8 KB, 4-way associative data cache (16-byte lines), 64-entry fully-associative instruction TLB, and 128-entry fully associative data TLB that are shared by the eight VPs. The eight SPARC physical cores are connected through a crossbar to an on-chip unified 4 MB, 16-way associative L2 cache (64-byte lines) which is banked eight ways to provide sufficient bandwidth for the eight SPARC physical cores.

3 Performance Results

3.1 Benchmark Performance for GAMESS on T2 Niagara Processor

We first benchmark *Luciferin* and *Ergosterol* molecules on a Niagara processor by running single jobs on different sets of VP combinations. We create VP sets such that the processes that are run on these VPs have access to as much hardware as possible for speedier execution. For example, if we need to create a set of 8 VPs, we distribute the VPs among all the 8 cores. We then bind the GAMESS processes to the VP sets that have been created so as to take advantage of the processor affinity property. Processor affinity [15] exploits the fact that some remnants of the process's state may remain in the processor's cache. The benchmarking results have been shown in Figures 1 and 2.

From the results we can deduce three different trends clearly. In case of the *direct* execution of *Luciferin* and *Ergosterol*, increase in the number of VPs used for execution results in better performance. The increase in hardware resources and the thread level parallelism help speed up the computations. This trend is not followed in case of the *conventional* execution of *Luciferin* and *Ergosterol*. The execution time of *conventional Luciferin* reduces initially untill about 32 VPs and then steadily increases as we move from 40VPs to 63VPs. *Conventional Ergosterol* degrades in performance as we increase the number of VPs. A *conventional* GAMESS job can be characterized into two parts. The first part is writing of the integral files and the second is RHF SCF calculation using the integral values that are stored in the disk files. The application fetches the integral values from the files and then performs the RHF SCF calculations iteratively. In case of *Ergosterol*, the integral files (size 22GB) cannot fit in the main memory of the Niagara processor (16GB). This gives rise to a large number of page faults and cache misses thus leading to a drop in performance. Another contributor to the slow execution time for the *conventional Ergosterol* molecule could be the issue with parallel reads and writes at higher thread counts that affect GAMESS [16] performance. This explains the degradation in the performance in a *conventional Luciferin* molecule once the number of VPs are increased beyond 32.

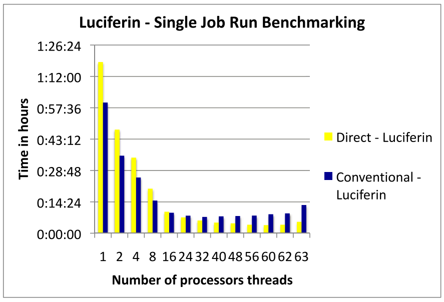


Fig. 1. *Luciferin*: Single job benchmark

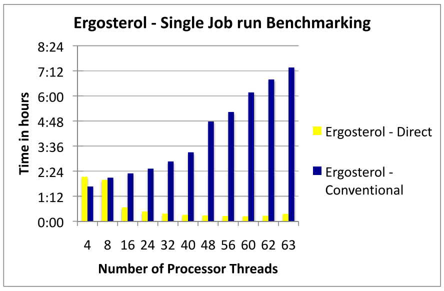


Fig. 2. *Ergosterol*: Single job benchmark

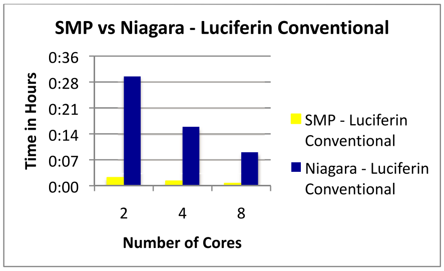


Fig. 3. *Luciferin*: *Conventional* job SMP Vs Niagara comparison

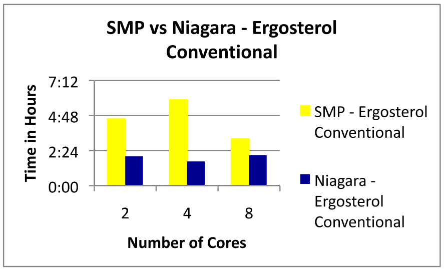


Fig. 4. *Ergosterol*: *Conventional* job SMP Vs Niagara comparison

One more thing to note is that the kernel itself is run on one of the 64 VPs. When we create processor sets, we cannot assign all the 64 threads to different processor sets since at least a single VP is required for running the kernel. In such a scenario, if we assign 63 VPs to execute a single GAMESS job, the job takes more time than when run with 60 VPs. Also, there is a steady increase in the execution time as we move from 60 VPs to 62 and 63 VPs as seen in the *direct* execution of *Luciferin* and *Ergosterol* in Figure 1 and 2. As the hardware resources used by the kernel are shared with the GAMESS threads, we can see a performance degradation. Hence 60 VPs would be an optimal number to be used for application usage and 4 VPs for the system usage.

3.2 Performance Comparison between T2 Niagara Processor and a SMP 8-Core Cluster

We compared the performance of GAMESS on a T2 Niagara processor with its performance on a SMP cluster. The SMP cluster contains 4 nodes, each containing two dual core Intel Xeon "Woodcrest" processors. For the sake of comparison with the Niagara processor, we used only two of the nodes on this cluster to run GAMESS (since this would be equivalent of running GAMESS

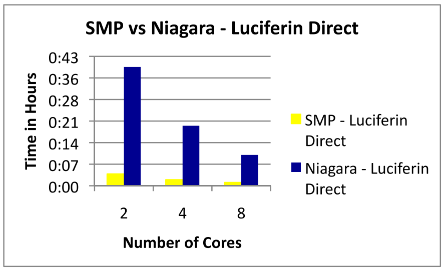


Fig. 5. *Luciferin*: *Direct* job SMP Vs Niagara comparison

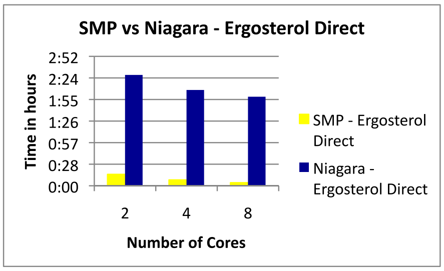


Fig. 6. *Ergosterol*: *Direct* job SMP Vs Niagara comparison

on 8 cores). The performance results for *conventional* and *direct* executions of *Luciferin* and *Ergosterol* are given in Figures 3, 4, 5 and 6.

We can see that the SMP out performs the Niagara Processor for the *Luciferin* molecule while it is worse than the Niagara performance in case of the *conventional* mode execution of *Ergosterol* molecule. We need to note that each core in an Intel processor can run only a single thread while each Niagara core can run up to eight simultaneous threads. The cache available to each core on an Intel processor chip is more than the cache available in a Niagara processor as the cache is distributed amongst all the cores. Each core on an Intel processor runs at a higher clock rate as compared to the clock rate of a VP in the Niagara processor. These clearly help to understand the reasons for the higher performance in case of the *Luciferin* molecule calculations and the *direct* calculations for the *Ergosterol* molecule.

However, the true advantage of using a Niagara processor can be seen when we run a *conventional Ergosterol* calculation. The main reason for this is the usage of a dedicated floating point pipeline in each of the 8 cores in a Niagara processor. The GAMESS calculations are inherently floating point in nature and dedicated floating point pipelines help to improve performance by nearly 50 percent. As indicated by the performance results, execution of a *conventional Ergosterol* calculation on the T2 processor is very time consuming, though it performs better than the SMP. The performance degrades as we increase the number of VPs on which GAMESS is run. This scenario can be readily exploited to improve the performance by utilizing the adaptation algorithm first introduced in [8]. The next section explains the suitability of this algorithm in a multi-core and multi-threaded scenario.

4 Adaptations in GAMESS Using NICAN Middleware

The SCF algorithm is one of the most computationally intensive parts in the GAMESS execution. Selection of the correct electronic structure calculation routine has a very big effect on the overall calculation and calculation time. The iterative nature of the SCF algorithm allows us to switch between the *conventional* and *direct* implementations in an arbitrary SCF iteration. The switching is carried out using the middleware tool NICAN in order to decouple the application from having to make any adaptation decisions during the application execution. The application is responsible only for the invocation of the adaptation handlers. The adaptations are handled by a control port that is part of the NICAN tool.

The adaptation scheme used in [8] for SMPs is summarized ahead. The adaptation scheme consists of a static and a dynamic part. Every *conventional* GAMESS job gets modified to a *direct* execution mode if there is a "peer" *conventional* GAMESS job already running in the system. It was shown in [9] that while running concurrent scattered GAMESS jobs, a single conventional job helps to achieve better performance. This constitutes the static adaptation method. The dynamic adaptation is used during the iterative SCF calculations. The control port gathers system and application information that allows it to decide on the adaptation at runtime using the algorithm given below.

```
$t_N$ = Actual time taken for iteration N
$t^u$ = Upper bound for the time per iteration (taken as a arbitrary large value)
$m$ = Average iteration time over N iterations
$t_0^e$ = Estimated ideal run time for running a single iteration (obtained by
NICAN after running a GAMESS check run at startup)
$\Delta t_0 = | t_0^e - t_0 |$
**if** ($t_i > t^u$ OR $t_i > m + \Delta t_0$) **then**
   **if** (SCF is *conventional*) **then**
      switch to *direct*
   **else if** ((no peer *conventional* jobs) AND (enough memory)) **then**
      switch to *conventional*
   **end if**
**end if**
```

The experimental results obtained for this algorithm on a SMP have been given in [8]. It has been shown on a two processor system with I/O congestion, that the performance of dynamically adaptive GAMESS is nearly the same as a "no-congestion" case. If the I/O bandwidth is fully consumed, then the adaptation scheme gives two times improvement in the execution time of GAMESS. Also, on running two simultaneous parallel GAMESS jobs on two and four processors, a gain of 10-15 percent in the cumulative execution time is obtained through a dynamic adaptation scheme. This dynamic adaptation algorithm holds true for a multi-core and multi-threaded environment as well. As seen in the benchmarking results of Section 3, the degradation in the performance of a *conventional Ergosterol* molecule calculation at higher values of VPs is a good starting point to apply the above adaptation algorithm.

4.1 Adaptation Results on T2 Niagara Processor

The adaptation scheme was tested by running simultaneous parallel GAMESS jobs on the T2 Niagara processor. The physical cores were partitioned equally among the jobs thus ensuring that the hardware resources of a core is used exclusively by the GAMESS job assigned to the VP belonging to that core. The performance was measured by executing two parallel GAMESS jobs that consisted of one *Luciferin* molecule and one *Ergosterol* molecule. Similar results were obtained for three parallel GAMESS job execution which have two *Luciferin* Molecules and one *Ergosterol* molecule. If the SCF method is not defined in the GAMESS input file, then GAMESS selects the SCF mode to be *conventional* by default. Hence both the above tests were performed using *conventional* SCF mode at the start. The performance graphs have been given in Figures 7 and 8. We have not distinguished between static and dynamic adaptation in the results.

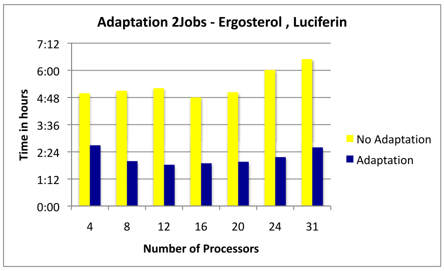


Fig. 7. Two simultaneous parallel jobs execution

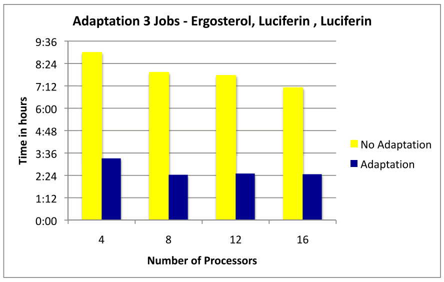


Fig. 8. Three simultaneous parallel jobs execution

We compare the performance of a non-adaptive GAMESS job ($GAMESS_{ORIG}$) and dynamically adaptive GAMESS ($GAMESS_{ADP}$) obtained using the NICAN middleware tool. We can see in the graphs that cumulative running time for $GAMESS_{ADP}$ is about 50 percent faster than $GAMESS_{ORIG}$. For two simultaneous GAMESS job execution, the adaptation gives a steady gain irrespective of the number of VPs used for running the simultaneous jobs. It was observed that at lower VP allocation values, the smaller molecule (*Luciferin*) is transformed into a *direct* method of execution due to the presence of a peer *Ergosterol* molecule but then switches back to *conventional* mode dynamically to ensure a faster run time. For larger VP allocations, both *Luciferin* and *Ergosterol* adapt and complete their execution in the *direct* mode. Similarly, in case of three simultaneous GAMESS jobs, we see that the cumulative run time of $GAMESS_{ORIG}$ is reduced by more than 50 percent by using the adaptation algorithm. All the three jobs complete their execution in the *direct* mode.

5 Conclusions and Future Work

The main focus of this work is to compare the performance of Electronic Structure calculations on a SMP with the performance on a T2 Niagara Processor. We have seen that SCF calculations for small molecules like *Luciferin* perform much better on a SMP than on the Niagara processor. This trend can be seen for the *direct* SCF calculation for an *Ergosterol* molecule as well. However, the Niagara processor provides much better performance as compared to a SMP when we consider a *conventional* execution of the *Ergosterol* molecule. The T2 Niagara processor looks to be a good processing environment for such an execution scenario.

We also demonstrated that by using the adaptation algorithm introduced in [8], we can obtain performance improvement in GAMESS on a multi-core and multi-threaded environment. We have shown that the execution of adaptive GAMESS can be several magnitudes faster than the non-adaptive GAMESS execution. On a multithreaded processor like the Niagara, the I/O becomes a bottleneck as we start increasing the number of threads for a *conventional* mode of execution. In such cases, it was observed that the *direct* mode is the best way of execution. The performance difference between the *conventional* and *direct* mode at higher allocations of VPs is essential in getting the adaptation to work on such processors.

As a future work, it would be interesting to see how the application adaptability behaves when we use a cluster of such multithreaded processors. We would like to develop multiple adaptation control strategies for usage on such processors. These could include strategies such as changing the thread allocation dynamically from a single core to span multiple cores and to take advantage of processor affinity. This requires further research into the performance of GAMESS at different VP allocation configurations, along with the cache and IO performance of the Niagara processor for GAMESS. It would also be interesting to examine how any other cluster application would behave on a Niagara processor. Since the GAMESS version used for testing uses TCP/IP as an underlying communication framework, suitability of using MPI applications can be explored and documented. Ultimately, the aim is to develop generic adaptation control strategies that can be reused with any parallel application and are scalable for a multi-core and multi-threaded environment.

References

1. Terboven, C., an Mey, D., Sarholz, S.: OpenMP on Multicore architectures. In: Chapman, B., Zheng, W., Gao, G.R., Sato, M., Ayguadé, E., Wang, D. (eds.) IWOMP 2007. LNCS, vol. 4935, pp. 54–64. Springer, Heidelberg (2008)
2. Kulkarni, D., Sosonkina, M.: A framework for integrating network information into distributed iterativesolution of sparse linear systems. In: Palma, J.M.L.M., Sousa, A.A., Dongarra, J., Hernández, V. (eds.) VECPAR 2002. LNCS, vol. 2565, pp. 436–450. Springer, Heidelberg (2003)
3. White, E.H., Capra, F., McElroy, W.D.: The Structure and Synthesis of Firefly Luciferin. J. Am. Chem. Soc. 83(10), 2402–2403 (1961)
4. Hennessy, J.L., Patterson, D.A., Arpaci-Dusseau, A.C., et al.: Computer architecture: A quantitative approach, 4th edn. Morgan Kaufmann, San Francisco (2006)

5. Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S., Windus, T.L., Dupuis, M., Montgomery, J.A.: General Atomic and Molecular Electronic Structure System. Journal of Computational Chemistry 14, 1347–1363 (1993)
6. Sosonkina, M.: Adapting Distributed Scientific Applications to Run-time Network Conditions. In: Dongarra, J., Madsen, K., Waśniewski, J. (eds.) PARA 2004. LNCS, vol. 3732, pp. 747–755. Springer, Heidelberg (2006)
7. Sosonkina, M., Storie, S.: Parallel performance of an iterative method in cluster environments: an experimental study. In: Proceedings PMAA 2004 (October 2004)
8. Ustemirov, N., Sosonkina, M., Gordon, M.S., Schmidt, M.W.: Dynamic Algorithm Selection in Parallel GAMESS Calculations. In: International Conference Workshops on Parallel Processing (ICPPW 2006) (2006)
9. Ustemirov, N., Sosonkina, M., Gordon, M.S., Schmidt, M.W.: Concurrent Execution of Electronic Structure Calculations in SMP Environments. In: Proceedings HPC 2005 (April 2005)
10. Ustemirov, N., Sosonkina, M.: Efficient Execution of Parallel Electronic Structure Calculations on SMP Clusters. Minnesota Supercomputing Institute Technical Report umsi-2005-227, University of Minnesota (2005)
11. Kongetira, P.: A 32-way Multithreaded SPARC(R) Processor. In: Proceedings of the 16th Symposium On High Performance Chips, HOTCHIPS (2004)
12. McDougall, R., Mauro, J.: Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture. Prentice-Hall, Englewood Cliffs (2006)
13. Olson, R.M., Schmidt, M.W., Gordon, M.S., Rendell, A.P.: Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model. In: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, November 15-21, 2003, p. 41 (2003)
14. Sun Microsystems Inc., http://www.sun.com/processors/UltraSPARC-T2/
15. Kazempour, V., Fedorova, A., Alagheband, P.: Performance implications of cache affinity on multicore processors. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 151–161. Springer, Heidelberg (2008)
16. Wu, M.-S., Bentz, J.L., Peng, F., Sosonkina, M., Gordon, M.S., Kendall, R.A.: Integrating Performance Tools with Large-Scale Scientific Software. In: IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007 (2007)

A Fuzzy Logic Fish School Model

Juan Carlos González, Christianne Dalforno, Remo Suppi, and Emilio Luque

Computer Architecture and Operating System Department (CAOS)
University Autonoma of Barcelona, Spain
{jgonzalez,chris}@caos.uab.es,{remo.suppi,emilio.luque}@uab.es

Abstract. This paper summarizes our work on fuzzy modeling for an Individual-oriented Model (IoM). Our model is particularly geared toward simulating the movement of fish schools, derived from the model by Huth and Wissel. The background and motivation for the problem as well as a description of biological model are given here. A fuzzy logic implementation is discussed based on the mathematical model proposed. Finally, the experiments performed to demonstrate that our model represents the real behavior of fish schools. Keywords: Fuzzy Logic, Individual-oriented Models (IoM), Fish School.

Keywords: Fuzzy Logic, Individual-oriented Models (IoM), Fish School.

1 Introduction

Individual-oriented Models (IoM) were created to solve to the limitations imposed by models based in a population that use differential equations, for example to explain the behavior of a group of individuals.

In IoM the unit of the system is the individual, so the group's heterogeneity can be represented through the differentiated behavior of each individual. Therefore, "individuals are the building blocks of ecological system. The properties and behavior of individuals determine the properties of the systems they compose" [1]. The system is not directly modeled as a globally integrated entity, individuals are autonomous.

In [2] many examples of IoM applications are given and the author outlines the importance of this kind of model for researches where the individuals' interactions are important to accurately represent the group's features. "Organisms get together and form populations and societies, have properties none of which can be explained by properties of the individual organism alone" [3].

We chose the simulation of fish schools as a case study. Some kinds of fish present the characteristically behavior of being together for a long time maintaining a self-coordinated movement without the presence of leaders [4]. This behavior caught the attention of the biologists interested in investigating this kind of formation.

The April 2007 issue of National Geographic [5] presents an overview of the situation of worldwide fishing. In fact, fish consumption has been growing and, in approximately 50 years, fishing went from about 30 million tons to nearly 100 million tons.

One of the species that has been suffering a lot due to an increase of its consumption is blue tuna. The situation of this specie is so critical that the International Commission for the Conservation of Atlantic Tunas (ICCAT), responsible for managing the reserves of blue tuna, was created. Equilibrium between commercial interests and preservation of the species must be found. Thus current research works focus their interest in fish behavior, reproduction, development and exploration. There are some models that represent the behavior of these populations [6], [7], [8]. The one used in this work is an Individual-oriented Model (IoM) [4]. This kind of model represents the behavior of a group by applying some rules to a group of individuals interacting among each other.

Our general interest is the application of fuzzy logic (FL) to simulations based on IoM. Within fish school simulations several models have been developed using conventional mathematical modeling as in [9], [10]. We use the fuzzy approach to achieve an alternative way of modeling. Also, by using fuzzy logic we odd a degree of uncertainty, which is present in reality, to the model.

In this work we start by using the biological model described in [4]. This biological model defines the behavior of each fish as a consequence of its interaction with four selected neighbors. A fish changes its reaction direction according to its neighbor's proximity. So in each simulation step each fish will shift to new position and direction in the simulation space. A qualitative scheme is defined through fuzzy modeling for represent the biological model. A systems behavior described in a natural language can be interpreted using this kind of scheme.

This paper is organized as follows. In section 2, we present the necessary background on the biological model that describes the fish interaction schemes. In section 3, we describe our implementation of the fuzzy model. We describe the experimentation made in section 4 and conclude the paper in section 5.

2 The Biological Model

Fish schools are one of the most common social groups in the animal kingdom. Many fish species are organized in groups without a hierarchical structure. These schools show a high level of synchronization in short and long periods of time. Therefore, the movement of a school of fish presents highly parallel orientation.

The biological model implemented in this work is based on an IoM developed by Huth and Wissel [4]. This model defines the behavior of a fish school representing simple rules to be applied to each individual. So the behavior of individuals and their interaction is what determines the dynamics of the group.

The organization of schools is based on a passive communication; the patterns of detection are their eyes and lateral line. Each fish observes the movement of its neighbor using its eyes and or its lateral line. Only in special situations do other senses take part in the communication.[11]

2.1 Interaction Areas

Each fish in the group changes its direction at each time step. This new direction depends on the position and direction of a fixed number of neighbors. So the influence of a neighbor depends on its relative position. We distinguish three different influences: attraction, parallel orientation and repulsion [4]. Fig. 1 shows three areas for each influence. These areas are bounded by three concentric circles and anything outside of R3 is considered the search area which has no influence in interaction. Additionally, the region defined by angle ω determines the null vision area or DA (Dead Area).

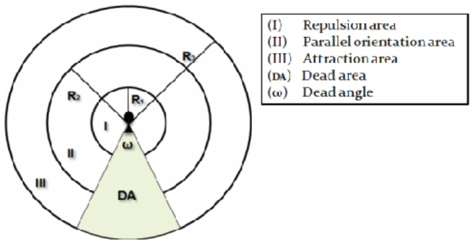


Fig. 1. Interaction Areas

Fish reaction depends on the area in which a neighbor is positioned. The areas are determined by the radii R1, R2, R3 and the dead angle ω. Typical values for these radios are: 0.5BL, 2BL, 5BL (Body length) and 30º respectively.

Repulsion occurs when the chosen neighbor is in range I. When two fish are very close, they tend to change their course to avoid collision. A neighbor found in range II influences the fish to move in parallel with it. It is a way to maintain a regular distance between them that is a parallel orientation situation. When a neighbor in range III is chosen, it attracts the fish to it. The attraction reduces the distance between fish maintaining the group's cohesion. Finally, the DA is the area behind the individual where no neighbors can be perceived.

The final reaction of the fish will be a combination of each reaction to the selected fish. When there are no neighbors in these areas, the fish starts looking for neighbors by swimming in random directions [7].

3 Fuzzy Model of the Fish School

The importance of our model is the use of qualitative modeling based on fuzzy logic. With the use of fuzzy logic is possible to approximate reasoning based on vague or incomplete knowledge. Therefore, there is nothing vague or fuzzy about fuzzy logic. It is natural and simple mathematics, which allows handling vague on difficult to specify information or. The utility of fuzzy sets lies in their ability to model uncertain or ambiguous data, so often seen in real life [12].

3.1 Fuzzy Structure

Fuzzy inference system (FIS) consists of a fuzzification interface, a rule base, a database, a decision-making unit, and finally a defuzzification interface [13]. A FIS with five functional blocks is shown in Fig 2.

FIS operation is described as follows. The crisp input is converted into fuzzy by using a fuzzification method. After fuzzification the rule base is formed. A rule base containing a number of fuzzy IF/THEN rules and a database which defines the membership functions of the fuzzy sets used. The rule base and the database are jointly referred to as the knowledge base. A decision-making unit which performs the inference operations on the rules. Defuzzification is used to convert a fuzzy value to a real world value which represents the output.

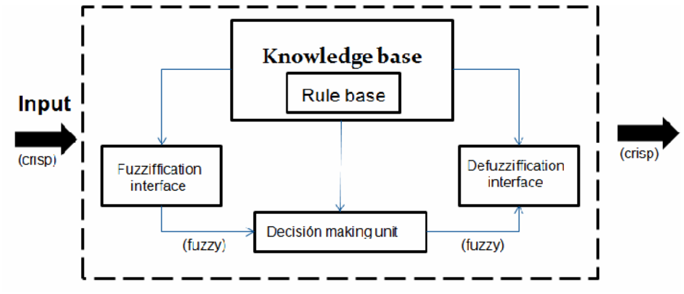


Fig. 2. Fuzzy Inference System

3.2 Model Structure

Our model consists of a fish school that moves inside an area where each fish is a unit of the system. The interaction between the fish results in the movement of the group. We define the area as a finite bidimensional space limited by an imaginary borders, where these borders also represent an external influence on the fish. To simulate movement, we have taken the location of each fish represented by x and y coordinates. The fish's speed and direction are considered.

Our model is derived from the model [4], described in section 2. We have made some assumptions which describe the interactions among individuals, as well as the numerical data that this has taken. First, it is necessary to mention the assumptions extracted from the biological model for each fish in order to explain how we are implementing it using fuzzy logic. These assumptions involve the behavior patterns that depict the movement of the fish school. Therefore, every fish:

– Selects its neighbors according to certain remote regions.
– Calculates the reaction to each of its neighbors or the external influence.
– Calculates the final effect based on the influence of all the reactions with its neighbors.
– Calculates a new position.

Starting from these assumptions we built a number of fuzzy systems to represent the model. Hence, we implemented three fuzzy systems; Proximity, Reaction and Aggregation. For these systems some variables such as position and orientation were defined and used. Within fuzzy logic these variables are called universe of discourse and they are represented by fuzzy sets.

3.3 Fuzzy Systems Description

We started by building a single fuzzy system; Proximity. This first system helped us represent how the fish identifies the distance to its neighbor. Therefore, we determined proximity as a fuzzy system to find the relative position with respect to its neighbor. By getting the position of an individual, the position of its neighbor and the orientation of the individual.

We took three variables for this system; the x and y distance and orientation. We defined distance as the gap that separates a fish from its neighbor taking into account the x and y axes. Orientation is the variable in which we can represent how the fish recognizes its nearest neighbors according to its area of vision. Also we determined if a fish has a neighbor located in the region of null vision (Dead area) or not. The output variable of the system is the consequence, of three input variables and gives proximity as a result.

Once a fish recognizes its four most nearby neighbors according to proximity, we built another fuzzy system called Reaction. This system represents the response of fish to the influence given by its neighbors in accordance with interaction zones. As a result, we have variables related with the orientation of its neighbor and itself. Hence, the output of this system represents the rotation a fish is able to do.

We also included an influence given by an external agent. This agent is an imaginary border that limits the area. Hence, if a fish is found very near to the border, it assumes the reaction of repulsion, which is a similar situation to when the fish has a neighbor in the zone I, i.e., it will change the orientation in order not to touch the border.

Once we have the four influences, there is a relation between them. In order to depict this, we built a third fuzzy system named Aggregation. Our scheme allows finding the final slew that the fish must do due to influences. This is done by linking pair of influences until the four reactions. Here, the influences are taken as vectors. i.e., our fuzzy system is implemented as a sum of vectors but the values are approximated. System output permits to determine a new position and orientation for each fish.

Now that the fuzzy systems have been defied, we describe the knowledge base that is the main part of the FIS used to build fuzzy systems. As we mentioned in section 3.1, the knowledge base is formed by a Rule base and a database, where the membership functions and the rules of all the system are establish.

Knowledge base. In all fuzzy sets, we determined a set of the fuzzy rules which involves behavior patterns of whole system according to assumptions mentioned above. Then using the knowledge of the biological system, we built a database in which we defined the membership functions of each fuzzy set. This is used in the process of fuzzification. The purpose of fuzzification is to map the inputs from a set of features to values from 0 to 1 using a set of input membership functions thus forming the fuzzy sets. Figure 3 shows the membership functions of each input for the Proximity fuzzy system. There are three inputs and an output: distance X, distance Y, orientation and proximity. These input membership functions represent fuzzy concepts such as "near" or "far," "very to the left" or "very to the right," "northwest" or "northeast". We included in this system; 8 fuzzy sets to represent the distance variables, 4 sets for orientation variable and 7 sets for proximity variable.

Once we defined all fuzzy sets, we continue to describing the rules. Each rule is formed by a precedent and a consequent. The precedents express an inference or inequality and the consequent is how it can be inferred, and represents the output. Therefore the sentence which an FIS relates and consistent in rules, is given by: IF precedent, THEN consequent. The number of rules of a system depends of the equation 1, which follows:

$$\#Rules = \#FS_1 \_ \#FS_2 \_ \#FS_3 \_ \ldots \#FS_n \qquad (1)$$

Where FS is the amount of fuzzy sets of a variable and n is the number of variables. We built 256 rules on Proximity system so we formed the base of the knowledge. E.g., one of these rules is: "If Distance X is *Far Behind* AND Distance Y is

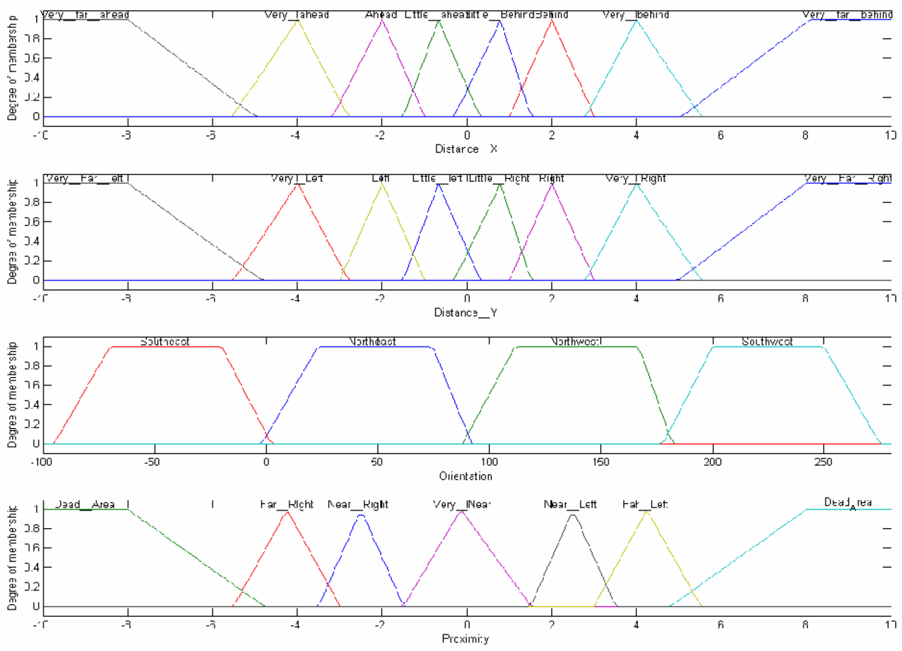


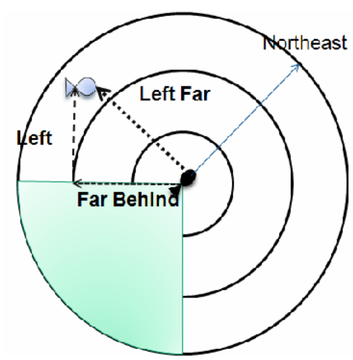
Fig. 3. Membership Functions to Proximity system

Very Left THEN Proximity is *Far to the Left*". This rule is represented in the figure 4 and it abstracted of Rule base of Proximity system. The same figure shows as is choose this rule starting of a rule sets. Figure 4a shows how a fish calculates the proximity with its neighbor and figure 4b shows how a rule is used to select an output given certain inputs.

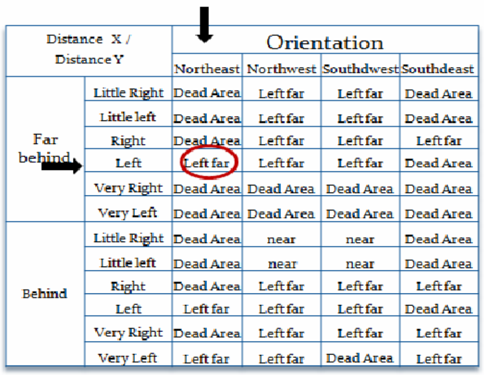
The Reaction fuzzy system is represented by fuzzy sets with request to the orientation and rotation. Within orientation, we have stretched the sets used on previous system; we now have 12 fuzzy sets. It means, the fish have 12 directions in which it can be facing. Figure 5 shows the orientation divided fuzzy sets. These fuzzy sets have a range from –90º to 270º. Also, a rotation variable is used as the system output. Here, we built 9 fuzzy sets to depict several turns. These sets can have linguistic variables such as: "Very large positive turn,""Large positive turn," "Medium positive turn," "Short positive turn," "null turn" similarly for "negative turn". These fuzzy sets have a range from –90º to 90º.

Finally, on the Aggregation fuzzy system uses the output variables as its inputs. Hence, we utilized the same fuzzy sets.

In terms of the rules, we built the rules based on the relationship of the four influences. But it should be noted that we relate these in pair of influences to produce a final effect. This is made in order to have a feedback system but with few variables. E.g., we compare two reactions as "large positive turn"' and "short negative turn". The union of these two reactions result another reaction. Then we compare the third reaction to this new reaction and then with the forth, similarly.



(a) Scheme proximity



Distance X / Distance Y		Orientation			
		Northeast	Northwest	Southdwest	Southdeast
Far behind	Little Right	Dead Area	Left far	Left far	Dead Area
	Little left	Dead Area	Left far	Left far	Dead Area
	Right	Dead Area	Left far	Left far	Left far
	Left	Left far	Left far	Left far	Dead Area
	Very Right	Dead Area	Dead Area	Dead Area	Dead Area
	Very Left	Dead Area	Dead Area	Dead Area	Dead Area
Behind	Little Right	Dead Area	near	near	Dead Area
	Little left	Dead Area	near	near	Dead Area
	Right	Dead Area	Left far	Left far	Left far
	Left	Left far	Left far	Left far	Dead Area
	Very Right	Dead Area	Left far	Left far	Left far
	Very Left	Left far	Left far	Dead Area	Left far

(b) Rule sets

Fig. 4. Proximity Fuzzy system

These sets are used to built rules according to interaction zones as well as the external influence. Therefore, according to this influence we designed a set of rules that represent a turn.

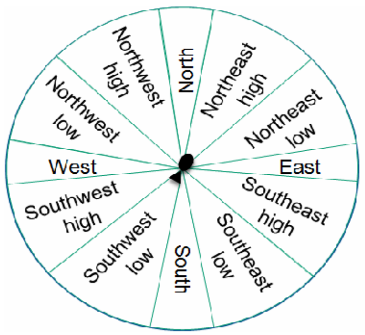


Fig. 5. Fuzzy sets of Orientation

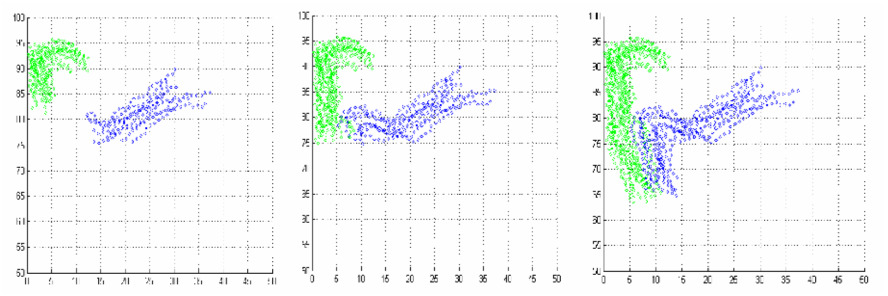
Finally, we defuzzify the output to obtain the new position of the fish that is calculated with the current position and final orientation. This process is repeated for each individual fish.

4 Experiments and Results

The correct functioning of the model can be analyzed at two levels. One aspect is to verify that our simulated fish will reproduce certain typical maneuvers. [14]. the second aspect is a measure of certain parameters of the global structure. These two ways of experimentation have been from the work presented by Huth and Wissel [4].

At a group level we compared the typical behavior of a real fish school with our simulation. This experiment is intended to describe the merging of two similar schools. When two schools meet in open water, the new swimming direction appears to be approximately the resultant vector of the two original tracks. In our experimentation we used two groups of 15 fishes for each shoal.

The result is shown in the Figure 6. Then of 10 steps of simulation, there are formed two schools. After 20 more iteration, the two schools are met and later they form an only school with same orientation.



(a) After 20 iterations (b) After 40 iterations (c) After 60 iterations

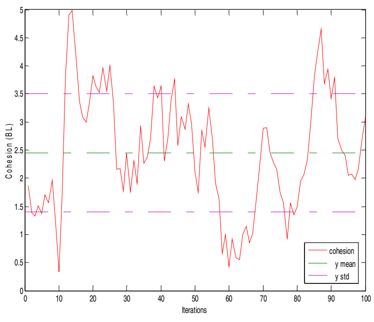
Fig. 6. Simulation of the merging of two schools

Also, the global structure of fish schools can be measured by two quantities: the nearest neighbor distance *nnd* and the polarization *p*. The nearest neighbor distance characterizes the typical distances in a school. It is defined as the average of the distance of every fish to its nearest neighbors. Typical values from the literature are around of 0.5-2 body length (BL).

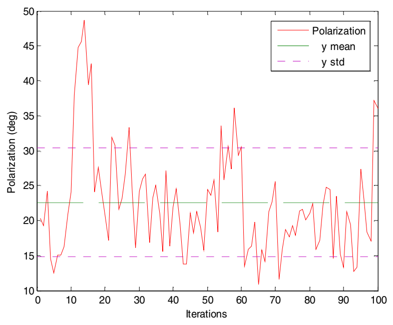
The polarization characterizes the intensity of orientation in the school. The polarization is defined as the average of the angle deviation of each fish to the mean swimming direction of the school. Schools with a high parallel orientation have polarization values between 10º and 20º.

A fish school with 15 individuals during 100 iterations was simulated to measure cohesion and polarization. These parameters are shown in the Figures 8a and 8b. From those parameters we calculate the mean and standard deviation of all iterations. In this simulation the means are 2.4 BL in cohesion and 22.2 en polarization. These results are found within of the typical values. Our simulations show variations on the cohesion and polarization of fish school along the time due to uncertainty given by the fuzzy sets. These results were what we expected.

We can see in the Fig.8a the cohesion of the group along of 100 iterations what it show some time steps in which the group presents compactness or high cohesion (σ<2BL) and low cohesion (σ > 4BL), e.g. Between 10th and 15th iteration the school presents low cohesion and at same time a confuse grade (ρ>60). This confuse behavior is due that the school is influenced by external agents as in this case the border (see Figure .6a).



(a) Cohesion



(b) Polarization

Fig. 7. Parameters measurement of the Fish School movement

5 Conclusions

In this work we proposed the usage of fuzzy modeling to develop Individual oriented models aiming to give some uncertainty to the individual behavior. As a case study we worked with fish schools behavior, an example of a biological system that has many known models cited in the best references but none using fuzzy logic. Hence, we assume that the way the fish realizes the distances between each other has been simulated and their decisions based on that data are not exact.

Our implementation is based in a biological model that that defines a set of general rules that we extended to a more detailed one. We built several fuzzy systems to represent this rules set: a system of proximity, a system of reaction and aggregation. These systems together represent the orientation of the fish school. To validate our simulator we proceeded in the same way that other authors have done: we compared the simulation results with the results of the original model [4].

The implementation using MATLAB did not present good performance, requiring a lot of time to simulate big populations. So, most of the time simulating we worked with few individuals to test and refine the simulator. Some studies show that the parallelization of the simulator can improve the system performance a lot [15],[16]. In future work we intend to parallelize this simulator to improve speed up and scalability. We are currently working on this version to extend the model to represent a more detailed system taking into account the presence of other species, predators and vegetation.

References

1. Grimm, V.: Individual-based modeling and ecology 4, 541–550 (2005), `http://jasss.soc.surrey.ac.uk`
2. Breckling, B., Middelhoff, U., Reuter, H.: Individual-based models as tools for ecological theory and application: Understanding the emergence. Ecological Modeling 194, 102–103 (2006)

3. DeAngelis, D.L., Rose, K.A.: Which individual-based approach is most appropriate for a given problem? In: DeAngelis, D.L., Gross, L.J. (eds.) Individual- Based Models and Approaches in Ecology, pp. 67–87. Chapman & Hall, London (1992)
4. Huth, A., Wissel, C.: The simulation of the movements of fish schools. J. Theor. Biol. 156, 365–385 (1992)
5. Montaigne, F., Warne, K., Carroll, C.: Crisis mundial de la pesca. National Geographic 20 (2007)
6. Anderson, J.: A stochastic model for the size of fish schools. Bull. Fish 8, 109–132 (1980)
7. Aoki, I.: Internal dynamics of fish schools in relation to inter-fish distance. Bull. Jap. Soc. Scient. Fish. 48, 1081–1088 (1984)
8. Parrish, J., Viscido, S., Grunbaum, D.: Self-organized fish schools: an examination of emergent properties. Biol. Bull. 196, 397–454 (2002)
9. Niwa, H.S.: Self-organizing dynamic model of fish schooling. J. Theor. Biol. 171, 123–136 (1994)
10. Hubbard, S., Babak, P., Sigurdsson, M.K.: A model of the formation of fish schools and migrations of fish. Ecological modeling 174, 359–374 (2004)
11. Shaw, E.: Schooling fishes critique and review. Devlopment and Evolution of Behaviour, 452–480 (1970)
12. Zadeh, L.: Itoward a theory of fuzzy systems. Aspects of network and system theory 8, 469–490 (1971)
13. Sivanandam, S., Smathi, S., Deepa, S.: Introduction to fuzzy logic using matlab, vol. 1, pp. 118–121. Springer, Heidelberg (2007)
14. Pitcher, T.J., Wyche, C.J.: Predator avoidance behavior of sand-eel schools: why schools seldom split. In: Noakes, D.L.G., Lindquist, B.G., Helfman, G.S., Ward, J.A. (eds.) Predators and Prey in Fishes, vol. 54, pp. 193–204. The Hague, Junk (1983)
15. Mostaccio, D., Dalforno, C., Suppi, R., Luque, E.: Distributed simulation of large-scale individual oriented models 2, 59–65 (2006), http://journal.info.unlp.edu.ar
16. Dalforno, C., Mostaccio, D., Suppi, R., Luque, E.: Increasing the scalability and the speedup of a fish school simulator. In: Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2008, Part II. LNCS, vol. 5073, pp. 936–949. Springer, Heidelberg (2008)

An Open Domain-Extensible Environment for Simulation-Based Scientific Investigation (ODESSI)

Adnan M. Salman, Allen D. Malony, and Matthew J. Sottile

Dept. Computer and Information Science, University of Oregon, Eugene, OR 97403 USA
{adnan,malony,matt}@cs.uoregon.edu

Abstract. In scientific domains where discovery is driven by simulation modeling there are found common methodologies and procedures applied for scientific investigation. ODESSI (Open Domain-extensible Environment for Simulation-based Scientific Investigation) is an environment to facilitate the representation and automatic conduction of scientific studies by capturing common methods for experimentation, analysis, and evaluation used in simulation science. Specific methods ODESSI will support include *parameter studies*, *optimization*, *uncertainty quantification*, and *sensitivity analysis*. By making these methods accessible in a programmable framework, ODESSI can be used to capture and run domain-specific investigations. ODESSI is demonstrated for a problem in the neuroscience domain involving computational modeling of human head electromagnetics for conductivity analysis and source localization.

1 Introduction

Computational science is now accepted as an important approach for scientific investigation, broadly considered equivalent in its discovery power to theoretical and experimental science. It is typically conducted through mathematical modeling and scientific simulation, leveraging access to advanced, high-performance computers (HPC) to run computational experiments (simulations) that seek to model reality in various domains. The evolution of computational science reflects both a growing need for computational power and increased sophistication of simulation methodology. Early concerns were on access to sufficient HPC resources, motivating research in parallel computing, computational grids, and large-scale storage. More recent research work in computational portals and workflows attempts to simplify resource access as well as provide programming support for coordinating simulation and analysis tasks. With computational horsepower becoming more ubiquitous, there is now growing interest in enhancing the discovery process of scientific investigations. In general, how productivity in computational-based science can be improved in practice will depend greatly on software environments that raise the level of investigation creation, execution, and management.

In scientific domains where discovery is driven by simulation there are common methodologies and procedures. An environment that can capture the shared standard practices and support their reuse across domains could improve productivity in scientific investigation creation and application. Methods such as parameter studies and tuning, optimization, uncertainty and sensitivity analysis, are generally used across simulation fields. Application of these methods in simulation studies typically require executing the simulation many times with different input parameter sets and data files. The environment could capture the common scientific methods in modules that can be contextualized for domain-specific use. The modules would hide the details of backend execution (implemented by the environment infrastructure), while providing an interface for their programming as part of an investigation workflow. The environment could also support other aspects of scientific investigations, including the management of input and output data, the specification of parameters, the post-processsing of results, and the generation of reports. The benefit is to provide a high level of service and automation to the computational scientist to enhance their work throughput and management.

In this paper we describe our research work to create and apply an environment for supporting scientific investigation called ODESSI (Open Domain-extensible Environment for Simulation-based Scientific Investigation, pronounced "Odyssey"). The environment will facilitate the representation and automation of scientific studies by capturing shared methods for experimentation, analysis, and evaluation used in simulation science in a framework that can be programmed and specialized for domain investigations. ODESSI will be demonstrated for scientific studies in the neuroscience domain involving computational modeling of the human head.

Section §3 describes the ODESSI objectives and design. The development of the ODESSI prototype is discussed in §4. ODESSI was inspired by our prior ICCS work [18,20] on computational modeling of human head conductivity. Section §5 outlines the domain problem in human brain science we are investigating and shows how ODESSI is applied to improve scientific productivity in this domain. Section §6 concludes with a discussion of research issues and outline of future research directions.

2 Related Work

The general theme of the ODESSI approach is to manage complexity in domain-specific scientific investigations by providing a programmable framework with high-level services for domain contextualization and use. Problem solving environments (PSE) are a traditional approach to addressing domain-relevant concerns by incorporating all the mathematical, algorithmic, and computational features necessary to solve a targeted class of science or engineering (S/E) problems [1,2]. The main goal of a PSE is to increase the productivity of scientists by letting them describe a problem and its solution in terms of the S/E concepts and use a highly-functional, integrated set of capabilities for modeling, analysis, and visualization. PSEs have been developed for partial differential equations (PDE) [3], linear algebra [4], chemistry [5], and other S/E areas. However, the traditional PSE approach has three important drawbacks: 1) it is difficult to create a new PSE, 2) PSEs are not developed to be reused, and 3) PSEs are hard to extend with new capabilities or new science methods.

One response to strict PSE design is to identify domain-level functionality that is common across related fields and build software tools that can be applied in developing computational science environments [6]. Scientific development environments take this idea further by offering rich components for data management and analysis, in a programming framework for scientific applications. For example, SCIRun [7] is a powerful environment for interactive computational science which has been used to create integrated problem solving environments in biomedical science [8]. ODESSI complements these directions by abstracting common simulation-based scientific methods in reusable components, providing a cross-domain framework for scientific investigation.

Grid computing and workflow systems research take a different tact by focusing on how to allocate and coordinate the use of computational resources (both systems and software/tool components) to create and run scientific applications such as GridLab[15]. Grid-enabled workflow systems such as Pegasus[9], Triana [10], and Kepler [11] are powerful tools being applied in computational science projects. However, their support for multi-experiment simulation workflows is still rudimentary and is not easily programmed for cross-domain use or execution on non-grid platforms. Web-based portals (e.g., the NEES [12] and BIRN [13] portals) and environments such as ViroLab [16] address some of these issues by offering higher-level S/E services (e.g., analysis, data management, simulation) while hiding backend complexity. The ability to abstract and reapply scientific methods for new scientific investigations or new scientific domains in these environments though is not supported well.

On the other hand, there are wealth of toolkits for scientific methods used in simulation. The DAKOTA toolkit [14] provides several optimization algorithms, uncertainty quantification, and parameter estimation. The Portable, Extensible Toolkit for Scientific Computation (PETSc) [17] is a suite of data structures and routines for the scalable (parallel) PDE-based scientific applications. The important aspect of these systems is their embodiment of a known scientific methodology in a programmable form. The idea behind ODESSI's approach is to provide a high-level scientific development framework that parameterizes and configures scientific methods for domain specialization.

3 ODESSI Requirements and Design

The goal of ODESSI is to provide a productive environment that assists domain scientists in the development and application of their computational investigations. To this end, the main requirements are:

1. To support common types of scientific methods used in simulation-based science.
2. To provide a programmable framework to contextualize methods for domain use.
3. To insulate the scientist from concerns of HPC resource usage, allowing them to focus on the process aspects of the domain investigation.
4. To provide record of simulation experiment for evolving scientific investigations.

The ODESSI environment shown in Fig. 1 was designed to support these requirements. The key concept of the ODESSI approach is the capture of standard procedures to conduct and analyze (simulation-based) scientific experiments in a modular, extensible, and reusable form. We call these procedures *scientific methods* and think of the methods as generating a set of simulation experiments to run. Common scientific methods include parameter studies, comparative analysis, optimization, sensitivity analysis, and uncertainty analysis. These methods are the basis upon which activities such as verification and validation, parameter tuning, and simulation-based experimentation are built for domain application. These processes that integrate different methods are the foundation of domain scientific investigations. A *scientific investigation* is a domain-specific

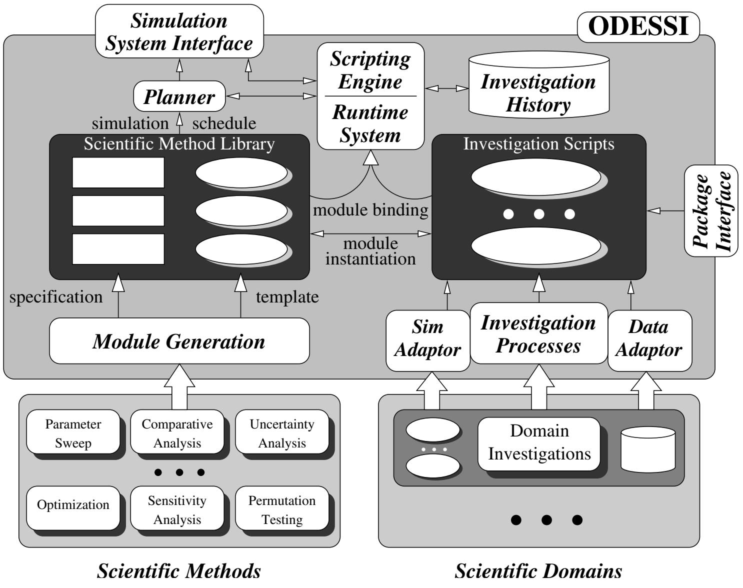


Fig. 1. Architecture and components of ODESSI framework

discovery process that applies one or more scientific methods in its lifetime. It defines the simulation codes to use, the input data files, and post-simulation analysis and visualization. If ODESSI can capture key scientific methods in easy-to-use modules, the level of productivity in the development and execution of scientific investigations may increase. We will focus our discussion on this aspect of the design.

Logically, ODESSI represents methods internally as *modules* consisting of two parts: a *specification* and a *template*. The specification identifies the context necessary for the execution of the modules, including the simulation program to be run and parameters. The template is the software construction of the module with abstract classes for operation of the specific scientific method. In this respect, the template embodies the method procedures for the generation of domain simulation experiments. A module is instantiated by an investigation script, setting the specification context and initializing the module state. When a method module is executed, it generates an experiment schedule that is passed to the ODESSI planner.

It is the responsibility of the ODESSI *planner* to conduct the necessary simulations on behalf of the invoked method. It is possible multiple methods are concurrently active, each with its own planner. The planner interfaces with the external simulation system to run a simulation experiment. It determines which experiments to execute based on the specified simulation schedule. If a method uses information from earlier experiments to determine future experiments, its module uses a dynamic schedule which is applied within the planner. The planner attempts to optimize schedules by interrogating the ODESSI *investigation history* to determine when simulation experiments have previously been conducted. A record is maintained in the ODESSI investigation history of every completed simulation experiment, containing complete metadata for the investigation and method specification.

4 ODESSI Development

In this section we describe the implementation of the conceptual design described in section §3. ODESSI is based on a set of Python objects that implement components shown in Fig. 1. These objects form a set of interacting threads that cooperate during the execution of the investigation script. An investigation script imports and instantiates the necessary objects that implement scientific methods that it uses, and these in turn interact through message passing operations once started. Simulation programs are invoked through the Python system interface.

An investigation script consists of three main sections. In the first section the user specifies the simulations under investigation and their initial input parameter values. The simulation specifications are used to create a simulation manager object that can be used to request simulation solutions. In the second section, the scientific investigation methods are customized, instantiated and launched for execution. Each scientific investigation method is executed in a separate thread. Each scientific method object derives from a base *Planner* object, adding the method-specific functionality that it is intended to provide. The third section is concerned with post-processing the results.

The ODESSI implementation uses messaging to communicate between threads. Due to the potential for long execution times during simulation runs, it is preferable to use an asynchronous execution model to allow threads to execute independently without blocking. Threads in ODESSI are based on the process and messaging model from the Erlang language [19] implemented by the Python "candygram" package. Each process has a message mailbox into which messages from other processes are delivered and later extracted and handled by the receiving process in the order of delivery.

ODESSI provides two entities that can be instantiated in the investigation script: the *Simulation Manger* entity and the *Scientific Investigation Method* entity. The Simulation Manager controls and manages the execution of a simulation. It acts as a server that provides solutions given sets of input parameters. Each instance of a simulation manger controls a single simulation. Multiple simulations can be controlled by multiple instances of the simulation manager. A simulation manager object can serve multiple requests from different threads. Scientific Investigation Methods provide the procedures that occur across scientific domains such as optimization and sensitivity analysis.

4.1 Scientific Investigation Methods

Scientific investigation methods are the common procedures that are used in several scientific domains. ODESSI currently supports optimization based on a parallel simulated annealing algorithm, simplex search, parameter studies and linear regression based sensitivity analysis. ODESSI can easily be extended with more methods and procedures. A scientific method gets executed in its own thread spawned by the main thread The scientific method is implemented as a module of four classes described below:

– A *specification* class providing a template to customize the investigation procedure.
– An *interface* class providing an interface for the user to instantiate and execute the scientific method. The interface class is parameterized with an instance of the method specification class and a simulation manager object. An interface object provides methods to start and stop execution and access results.

– The *scientific method logic* class implements the algorithm or the procedure of the scientific method and must inherit from the planner base class.
– The *planner* abstract class that all scientific method logic classes must inherit from. This class defines the interactions between the scientific method and the execution manager, cleanly separating scientific method logic from communication code.

4.2 Simulation Manager

The simulation manager class is the user interface to the execution manager class. A simulation manager object is created in the main thread.A simulation object and other optional parameters are passed in at initialization. At instantiation, the simulation manager spawns a thread and starts the execution manager. The simulation manager class currently provides methods to control execution and measure execution timing.

– *The execution manager:* The execution manager manages and controls a single simulation. It acts as a server that provides the simulation response given input parameters. The execution manager is an internal class and is not modified by the user. It is instantiated and run in its own thread by the simulation manager, and the planner base class requests solutions from the execution manager. Threads request solutions from the execution manager by delivering messages to its mailbox. In handling the request the simulation manager employs a dynamically sized pool of workers for simulation execution and a solution database manager.
– *Workers:* Each worker in the pool of an execution manager corresponds to the execution of a simulation on a resource. Workers interact with a simulation either through a very simple modification to the main function of the simulation code or through a wrapper around the unmodified simulation. The communication protocol between ODESSI workers and the simulation is very simple. Messages are defined for starting, stopping, parameterizing, and retrieving results from simulations.
– *The database manager:* The database manager implements the investigation history component of ODESSI and manages a repository where simulation solutions can be obtained. Solutions are associated both with the simulation code which produced them and the input parameters necessary for the run. This allows for both provenance tracking of simulation results and performance enhancement by avoiding redundant computations when the output already exists in the database.

5 ODESSI Application

ODESSI was inspired by our research in human neuroscience where we are developing computational models of human head electromagnetics for use in dynamic brain analysis [18,20,21]. The main goal of our research is to estimate the locations of the active brain regions given measured electroencephalogram (EEG) recordings. Called the *source localization* problem, its accurate solution will provide an opportunity to analyze cortex dynamics at high temporal and spatial resolution. The source localization problem has two parts:

1. *Forward problem*: given electrical sources (e.g., cortex dipoles), tissue geometries and conductivities, determine head volume and scalp electrical potentials.
2. *Inverse problem*: given an accurate forward solution, find optimal sources to match measured scalp potentials.

An accurate forward solver requires knowledge of the head tissues geometry (obtained from MR or CT images) and their conductivities. To determine the conductivities of the head tissues, we must solve the conductivity inverse problem. Here, a small current is injected in a subject's head and the response is measured on the scalp using electrical impedance tomography (EIT) technology. A search for optimal conductivities parameters can then be performed using the forward simulation compared to the measured potentials. Once the conductivities are found for an individual, a distributed dipole linear inverse solver can be built for EEG localization [22].

There are several challenges in this research. From the start, the source localization problem is ill-posed, since EEG measurements are made on (up to) 256 sensors and there may be thousands of cortex dipoles active. In addition, there are multiple sources of measurement error and modeling uncertainties that ultimately contribute to the accuracy of the solution as well as the performance. Measurement errors include the quality of MR/CT images, electrode and dipole registration, injected current level, and the EEG electrode data. These errors lead to modeling inaccuracies which propagate uncertainty in the solution results and also can affect computational efficiency. These include discretizing the PDEs, adjusting the computational grid resolution, and accurately segmenting the head tissues. Further, selection of parameters and modeling algorithm in the forward and inverse solvers also influence the final result.

How can we understand the quality of our source localization solutions and their use in dynamic brain analysis when dealing with multiple sources of measurement error and modeling uncertainties in constructing head models? Our desired scientific investigations involve computational processing to generate candidate models, as well as verification and validation to determine the effects of uncertainty and the robustness of solutions. In general, these objectives are shared with other scientific domains. In the following we outline the use of ODESSI in conducting several analyses from our domain, in particular showing the results from sensitivity analysis studies.

Conductivity Modeling. In our previous work we developed an inverse solver based on parallel simulated annealing algorithm to estimate the head tissue conductivities by solving the conductivity inverse problem. With ODESSI we were able to set up the problem and adjust the optimization parameters by only interfacing with the optimization method module. ODESSI made it trivial to experiment with different optimization objective function and different optimization algorithms.

PDF Solver Tuning. Our forward model is based on solving the time-dependent Poisson equation and considering the steady state solution as the static solution. The convergence of the forward solver depends, on two parameters. The *time step*, which controls the speed of reaching the steady state, and the convergence *tolerance* which specifies the level of accuracy. We used ODESSI to tune these parameters by performing a parametric study for different current injection pairs and different sets of conductivities.

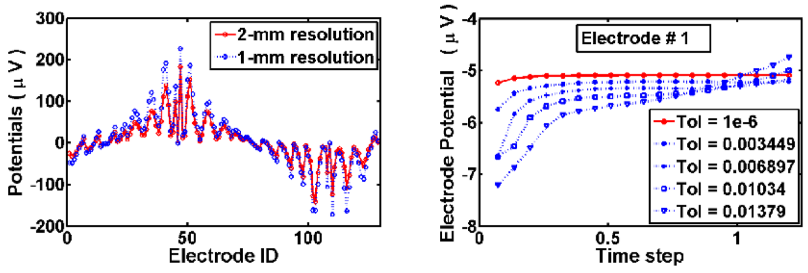


Fig. 2. Left, the potentials at the electrodes using $1mm$ resolution geometry vs. $2mm$ resolution. Right, tuning the forward-solver convergence parameters (time-step vs. tolerance).

Figure 2 shows a sample from this study. As the tolerance increases the solution fails to converge. For very small time step, the solver terminates prematurely.

Geometry Resolution Error. The geometry of the head tissues is obtained from imaging such as MRI or CT scans. Geometry obtained from high resolution ($1mm$) MRI captures more details about the head tissues, such as wholes in the skull. However, the computational time is significant. We use a high resolution image to construct lower resolution geometry by eliminating every other plane from the high resolution image. Then we used ODESSI to evaluate the error caused by this approximation. RDM and MAG metrics are used to compare between the solutions obtained using the two geometries. Our results show that the average RDM is about .8 and the average MAG is about .1. Therefore, the $2mm$ geometry can be used for visualization and testing. However, we have to use the high $1mm$ resolution to obtain accurate conductivity reconstruction. Here ODESSI allows us to experiment with the metrics for comparison.

Sensitivity Analysis. We further applied ODESSI for regression analysis to study how the uncertainty in each electrode potential can be apportioned to uncertainties in the inputs. In this analysis, we only considered the head tissue conductivities. A multivariate sample of 1000 points of the head tissue conductivities is generated. The conductivity of each head tissue is sampled from the normal distribution with mean equal to the average accepted value from the literature and the standard deviation is chosen such that the distance between the mean conductivity and the lower and upper bounds is about 3 standard deviations. Then the potentials at the electrodes are computed for each sample and a multiple regression fitting is performed on the standardized conductivities and electrode potentials. The standardized regression coefficients (SRC or β) quantify the effect caused by changes in the model independent variables from their mean values in terms of standard deviations.

Figure 3 shows distributions of the electrodes sensitivity due to changes in tissue conductivities. Positive β coefficients (SRC) correspond to electrodes near the current source while negative β coefficients correspond to electrodes near the sink. From the distributions we see that the potentials at all electrodes are insensitive to variation of the CSF tissue. This can be reasoned to the fact that the CSF tissue size is small and the variation in its conductivity is small. The second important observation is that the potentials are sensitive to changes of the brain conductivity. This observation contradicts

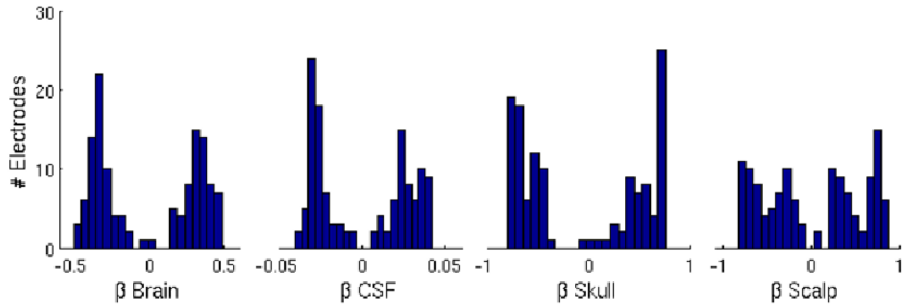


Fig. 3. Distribution of the electrode's sensitivity due to changes in tissue conductivities

the belief that most of the current will be shunted in the scalp. Therefore, we believe it is possible to explore the brain with EIT technology. We explain this sensitivity due to the fact that the brain is a big tissue and the wholes in the skull –which our forward solver captures– allow the current to go through the skull (contrary to spherical models). The third observation is that the potentials are highly sensitive to changes in the skull and scalp conductivities as expected, since the current sources are on the scalp.

Identifying and ranking the sensitivity of the electrodes due to model input variables are very important in our research. For instance, the conductivity of the CSF tissue can be considered homogeneous and be fixed at the literature accepted value. Also, the contributions of the electrodes potentials in computing the objective function can be weighted based on their sensitivity in the conductivity inverse problem.

Without the ODESSI framework, this detailed sensitivity analysis would not have been possible. We parameterized the sensitivity testing template in ODESSI and interfaced the simulation code. Once configured, the investigation required thousands of simulations to generate the results. These simulations were fully automated by ODESSI.

6 Conclusion and Future Work

ODESSI provides a framework for constructing scientific investigations by instantiating common methods for simulation-based analysis with domain-specific context. From a productivity perspective, ODESSI enables a (potentially large) number of simulations generated from a domain investigation processes to be run in a systematic and automated way. Extending our earlier ICCS research to allow investigation of uncertainty in brain conductivity and source modeling absolutely depended on this capability. The sensitivity results presented demonstrate the investigative power that can be achieved with relatively simple ODESSI programming.

However, the ODESSI concept extends further to domain support for data management (e.g., EEG and MRI data), results processing (e.g., statistics, data mining), visualization (e.g., plotting, 3D graphics), and meta-analysis. The key idea is how ODESSI can capture domain information to parameterize and contextualize common methods in these areas for high-level use. Our immediate goal is the development of the investigation history database to track the provenance of simulation experiments.

Acknowledgement

This research is supported by the Department of Defense, Telemedicine Advanced Technology Research Center (TATRC), grant number W81XWH-07-2-0092.

References

1. Gallopoulos, S., Houstis, E., Rice, J.: Computer as Thinker/Doer: Problem-solving Environments for Computational Science. IEEE Comp. Sci. and Eng. 1(2), 11–23 (1994)
2. Rice, J., Boisvert, R.: From Scientific Software Libraries to Problem-solving Environments. IEEE Computational Science and Engineering 3(3), 44–53 (1996)
3. Akers, R., Kant, E., Randall, C., Steinberg, S., Young, R.: Scinapse: A Problem-solving Environment for Partial Differential Equations. IEEE Comp. Sci. and Eng. 4(3) (1997)
4. Petitet, A., Casanova, H., Whaley, R., Dongarra, J., Robert, Y.: A Numerical Linear Algebra Problem-solving Environment Designer's Perspective. In: SIAM Annual Meeting (1999)
5. Extensible Computational Chemistry Environment, http://ecce.emsl.pnl.gov/
6. Cuny, J., et al.: Building Domain-Specific Environments for Computational Science: A Case Study in Seismic Tomg. Int. J. of Super. App. and High Speed Comp. 11(3) (1997)
7. Parker, S., Johnson, C.: SCIRun: A Scientific Programming Environment for Computational Steering. ACM/IEEE Conference on Supercomputing (1995)
8. MacLeod, R., et al.: SCIRun/BioPSE: Integrated Problem Solving Environment for Bioelectric Field Problems and Visualization. IEEE Int. Symp. on Biom. Img. 1, 640–643 (2004)
9. Deelman, E., et al.: Pegasus: Mapping scientific Workflows onto the Grid. In: Dubitzky, W., Schuster, A., Sloot, P.M.A., Schröder, M., Romberg, M., et al. (eds.) GCCB 2006. LNCS (LNBI), vol. 4360, pp. 11–20. Springer, Heidelberg (2007)
10. Churches, D., et al.: Programming Scientific and Distributed Workflow with Triana Services. Concurrency and Computation: Practice and Experience 18(10), 1021–1037 (2006)
11. Ludäscher, B., et al.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice and Experience 18(10), 1039–1065 (2006)
12. NEES Cyberinfrastructure Center, http://it.nees.org/
13. BIRN - Biomedical Informatics Research Network, http://www.nbirn.net/
14. The DAKOTA Project, http://www.cs.sandia.gov/DAKOTA/software.html
15. GridLab, http://www.gridlab.org/
16. ViroLab, http://www.virolab.org/
17. PETSc: Portable, Extensible Toolkit for Scientific Computation, http://www.mcs.anl.gov/
18. Salman, A., Turovets, S., Malony, A., Tucker, D.: Computational Modeling of Human Head Conductivity. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005. LNCS, vol. 3514, pp. 631–638. Springer, Heidelberg (2005)
19. Armstrong, J.: Programming Erlang: Software for a Concurrent World (2007)
20. Salman, A., Malony, A., Turovets, S., Tucker, D.: Use of Parallel Simulated Annealing for Computational Modeling of Human Head Conductivity. In: ICCS 2007, pp. 86–93 (2007)
21. Salman, A., Malony, A., Turovets, S., Tucker, D.: On the Role of Skull Parcellation in the Computational Modeling of Human Head Conductivity. EIT, pp. 16–18 (June 2008)
22. Pascual-Marqui, R.: Review of Methods for Solving the EEG Inverse Problem. Int'l. Journal Bioelectromagnetics 1, 75–86 (1999)

Pattern-Based Genetic Algorithm Approach to Coverage Path Planning for Mobile Robots

Muzaffer Kapanoglu, Metin Ozkan, Ahmet Yazıcı, and Osman Parlaktuna

Eskisehir Osmangazi University, College of Engineering,
Eskisehir, Turkey
{muzaffer,meozkan,ayazici,oparlak}@ogu.edu.tr

Abstract. Sensor-based mobile robot coverage path planning (MRCPP) problem is a challenging problem in robotic management. We here develop a genetic algorithm (GA) for MRCPP problems. The area subject to coverage is modeled with disks representing the range of sensing devices. Then the problem is defined as finding a path which runs through the center of each disk at least once with minimal cost of full coverage. The proposed GA utilizes prioritized neighborhood-disk information to generate practical and high-quality paths for the mobile robot. Prioritized movement patterns are designed to generate efficient rectilinear coverage paths with no narrow-angle turn; they enable GA to find optimal or near-optimal solutions. The results of GA are compared with a well-known approach called backtracking spiral algorithm (BSA). Experiments are also carried out using P3-DX mobile robots in the laboratory environment.

Keywords: Genetic algorithms, Coverage path planning, Mobile robot.

1 Introduction

Start-goal oriented path planning methods do not address the problem of complete coverage path planning, in which every point in a given workspace should be covered at least once [1]. Complete coverage is needed for a variety of applications such as floor cleaning, lawn mowing and street sweeping, and painting [2]. These applications require robot apparatus to move over all the points in the free workspace of the environment. On the other hand, landmine detection, foraging, and patrolling kind of applications require robot sensing range to cover all the points in the environment [3], so called sensor-based coverage.

Different coverage path planning algorithms are proposed for the former applications above ([4], [5], [6], [7]). In the approaches ([4], [5], [6]), the environment is modeled using squares proportional to robot's tool size. However, most of mobile robots use sonar-rings or laser range sensors as sensing devices. Since these devices have a circular sensing range, a disk-based modeling approach may be more effective. Therefore in this study we adopted disk-based modeling approach.

After modeling the environment, the next step is to determine an effective path for coverage. In [4], a method called backtracking spiral algorithm (BSA) is proposed using spiral filling paths built based on squares of robot's tool size. In [5], the workspace is divided into squares with dimension of four times of the robot's tool. Then a spanning tree based approach is used to generate the path. In [6], the path is planned to guide the autonomous agricultural equipment to completely cover a field while avoiding all known obstacles.

In this study, a pattern-based genetic algorithm is proposed for sensor-based coverage path planning using disk shaped modeling. To the best of our knowledge there are a few genetic algorithm-based coverage planning methods ([6], [7]) for the former applications, and there is no genetic algorithm-based method for the sensor-based coverage. In [8], a disk-based modeling has been studied with a single neighbor-disk prioritization pattern. However, performance degradation due to double-coverage has been observed for some obstacle layouts. In this study, the number of patterns is increased to eight to overcome double coverage problem.

Genetic algorithms utilize the Darwinian and Mendelian principles of genetic evolution. Following its proposal by Holland in 1975, GA's have achieved a high level of popularity owing to its success [9]. Especially, combinatorial and/or non-convex problems have significantly benefited this unorthodox way of problem solving. GAs work on the list of decision variables called chromosomes in which a solution can be obtained directly or by decoding. There are several works related to GA for autonomous mobile robots ([10], [11]) in the literature planning a path from a start position to a goal position rather than dealing with a complete coverage.

In the following section, the proposed algorithm is presented. The experiments and comparisons with BSA have been displayed for real environments with obstacles in Section 3. Conclusions are given in Section 4.

2 A Pattern-Based Genetic Algorithm for Mobile Robots

In this study, a pattern-based genetic algorithm utilizing the rectilinear moves represented by eight neighbor-disk prioritization patterns is proposed. The coverage area is divided into overlapping disks, such that, no redundant overlaps are generated and no area is left out of robot's coverage as in Fig.1. The radius of each disk equals to sensing range of the robot sensor which is much greater than robot's physical dimension.

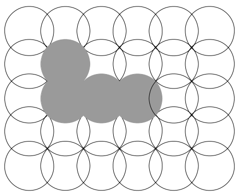


Fig. 1. Disk placement pattern

In Fig. 1, obstacles are shown with gray disks. Here we assume that a disk is either fully occupied or completely free. With this modeling, if the robot passes through the center of each disk, the environment would be fully covered. Therefore, the sensor based coverage problem turns into planning a path to visit all of the disk centroids. The objective of this approach is to minimize the multiple visits of disks while supporting rectilinear moves.

To determine the visiting order of the disks, a GA-based method is developed. Eight premeditated neighbor-disk-prioritization patterns (PP) (first four are given in Fig. 2) are designed to provide disciplined, reasonable rectilinear moves. The robot is relatively located in the middle of the each PP indicated by P-prefix in Fig.2. The numbers in the patterns indicate the preferred neighbor-disks. In other words, each PP prioritizes the neighbors in ascending order so that the disk numbered 1 has the highest priority if unvisited yet. 1-4 numbered disks are one-disk away from the robot's current disk, while 5-12 disks are two-disks away, and so on. Although 1-4 numbered disks are in the same distance, they are also prioritized according to the direction they represent. The PP will guide the robot until a dead-end is arrived. In this case, an unvisited neighbor disk at the shortest rectilinear distance is sought. If one found, the path is directed to that disk, by double-visiting at least one disk. The operation taking the robot from a dead-end to an unvisited disk is called *recovery*. Since one PP may guide the robot to a dead-end while another may never cause such a dead-end, different premeditated PPs are used to obtain sophisticated possible patterns for the path.

			13			
		24	5	14		
	23	12	1	6	15	
22	11	4	P1	2	7	16
	21	10	3	8	17	
		20	9	18		
			19			

			19			
		20	9	18		
	21	10	3	8	17	
22	11	4	P2	2	7	16
	23	12	1	6	15	
		24	5	14		
			13			

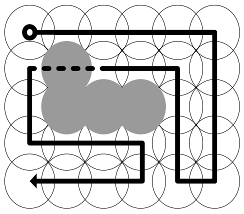
			13			
		14	5	24		
	15	6	1	12	23	
16	7	2	P3	4	11	22
	17	8	3	10	21	
		18	9	20		
			19			

			19			
		18	9	20		
	17	8	3	10	21	
16	7	2	P4	4	11	22
	15	6	1	12	23	
		14	5	24		
			13			

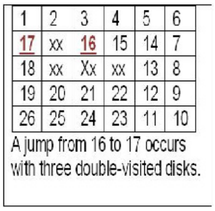
Fig. 2. Neighbor-disk prioritization patterns

Representation

In the proposed algorithm, a k^{th}*-neighbor* representation is preferred. Each gene represents the next neighbor's number as indicated by their driving PP. The order of genes in chromosomes (i.e., locus) corresponds to the move number. For an N-disk coverage problem, the length of the chromosome equals (N-1) genes corresponding to free disks. A sample environment is given in Fig. 3a. The environment is modeled by 30 disks: 4 of them are occupied by obstacles hence requiring (30-4-1=) 25 genes. Starting from the upper left corner and by using pattern P1, a possible coverage path is determined as in Fig. 3b. In this figure the number in a cell corresponds to the place of the gene in the chromosome. Since P1 assigns the first and second priorities to up and right moves respectively, and the up moves face the boundary, second priority move which is to the right is applied for next five disks. Following the same pattern, P1, the next four moves occurs at the third priority move which is downward leading to the bottom boundary. This operation continues till the robot faces a dead-end after 15 moves. First 15 moves leading to disk number 16 are considered regular pattern-based moves. At that point, a shortest path to an unvisited disk is determined. The disk number 17 has the 11th priority with respect to P1. For accessing nearest unvisited disk, shortest path procedure generates a path starting at disk 16 and ending at disk 17 and crossing over disks 1, 2, and 3 once again. Dashed lines in Fig. 3.a represent the shortest path between disks 16 and 17 via disks 1, 2, and 3. The remaining path is a result of P1. Encoded chromosome of this example is given in Fig. 3.c. In this figure, for example, the value of gene#1 is 2 representing right move.



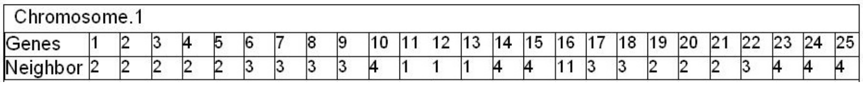
(a)



1	2	3	4	5	6
17	xx	16	15	14	7
18	xx	Xx	xx	13	8
19	20	21	22	12	9
26	25	24	23	11	10

A jump from 16 to 17 occurs with three double-visited disks.

(b)



Chromosome.1

Genes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Neighbor	2	2	2	2	2	3	3	3	3	4	1	1	1	4	4	11	3	3	2	2	2	3	4	4	4

(c)

Fig. 3. a) Sample environment, b) Decoded chromosome, c) Genetic representation

The chromosomes generated by using patterns increase the variety in the initial population with rectilinear moves. If the area contains no obstacles, GA will less likely improve the paths built based on pattern-driven moves. Patterns imply a relative location; therefore they must be evaluated for each disk when needed. As the number of PPs increases, the proposed pattern-based GA should be expected to improve paths with less number of redundant coverage at earlier generations. However, patterns included in this study are designed empirically leaving design, selection, and optimal number of PP to the future studies.

The steps of the proposed pattern-based GA are as follows:

```
Generate Initial Population;
While (stopping criteria are not satisfied)
    Evaluation & Selection
    Reproduction
    Crossover
    Mutation
    Enlarged New Population
Loop
```

These steps are explained in the following subsections.

Generation of Initial Population

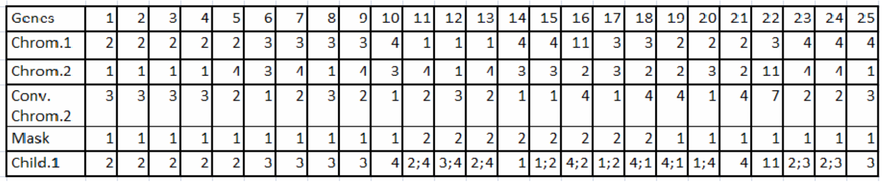
In building the initial population, all eight PPs are used to generate a variety of rectilinear movement patterns overcoming obstacles. The population size is divided as equally as possible among PPs. However, for increased variety, we perform a perturbation over PP, and then let the pattern-driven moves handle the rest. Population is first filled with chromosomes generated with respect to PPs, 8 chromosomes for this case. For generating the remaining chromosomes, these 8 chromosomes are manipulated by perturbing a gene's value corresponding to a feasible move randomly. These perturbations act as a mutation in the initial population. Main expectation from these pattern-perturbed moves is to let the path out from the local best solutions. When a pattern is interrupted by an obstacle, then a *Shortest Unvisited Neighbor procedure* is run the same way as its name implies. It means that there will be a *"jump"* from one disk to another indicating an inevitable repeated coverage of some disk(s).

Evaluation and Selection

The fitness value of a chromosome corresponds to the number of disks visited by the robot for complete coverage of the environment. Since an ideal solution contains no redundant coverage, the ideal fitness is equal to the distance of travelling each disk exactly once. Therefore the fitness of a solution increases by minimizing the number of disks visited more than once. Stochastic tournament selection with elitism is applied based on fitness values.

Crossover Operator

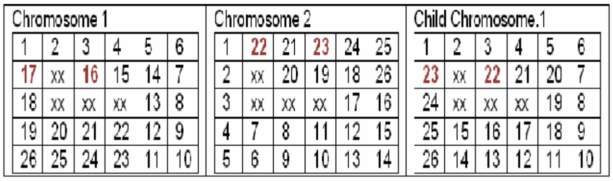
We customized parametric crossover for aligning with pattern-driven moves. A crossover mask is used with a masking probability. Masking probability determines the percent of the genetic material that will be exchanged between the chromosomes. If both chromosomes were built based on the same PP, this operator picks a random number for each gene. If this random number is greater than or equal to the masking probability then that gene is replaced with the gene value of the chromosome#1 if it leads to a feasible move. If the corresponding move is not feasible, then priority number is increased until a feasible move is reached. If the two chromosomes are generated with respect to different PPs, then one of the patterns must be expressed in terms of the other PP. The PP that is kept as is will be called the *reference pattern*. An example of crossover operator is illustrated in Fig. 4. In Fig.4, the P1 is used as the reference pattern. Chromosome 1 and 2 are selected for crossover operator. Chromosome 1 is generated with respect to P1, and chromosome 2 is generated with respect to P4. Therefore, the first priority disk of P4 corresponds to the 3rd disk of P1. For ensuring that each PP refers to the same disk during crossover operation, chromosome 2 is expressed in terms of reference pattern which is P1. While generating Child.1, we process chromosome 1 (Chrom.1 of Fig.4) and converted chromosome 2 (Conv. Chrom.2 of Fig.4). The row labeled Mask in Fig.4 represents if value is taken from the chromosome 1 or 2 depending on the random values as selected. This way, Child.1 takes value 2 in the first gene from the first chromosome depending on the mask value. If the selected random number were less than masking probability, the value of the first gene would turn out to be 3 which is the value of the converted chromosome 2. Another parameter related to this operator is crossover rate which determines what percent of the chromosomes will be subject to the crossover operation.



Genes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Chrom.1	2	2	2	2	2	3	3	3	3	4	1	1	1	4	4	11	3	3	2	2	2	3	4	4	4
Chrom.2	1	1	1	1	4	3	4	1	4	3	4	1	4	3	3	2	3	2	2	3	2	11	4	4	1
Conv. Chrom.2	3	3	3	3	2	1	2	3	2	1	2	3	2	1	1	4	1	4	4	1	4	7	2	2	3
Mask	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
Child.1	2	2	2	2	2	3	3	3	3	4	2;4	3;4	2;4	1	1;2	4;2	1;2	4;1	4;1	1;4	4	11	2;3	2;3	3

Fig. 4. Illustration of the modified parametric crossover operator for two chromosomes with masking probability

The impact of the crossover for the example environment is shown in Fig. 5. The path presented in Fig.5 is built based on the moves obtained from Child.1 chromosome of Fig.4. First 10 moves obtained from parents yield a feasible pattern. As a result of crossover of the parents, gene#11 of child chromosome 1 would take the value of 2. However this move is not feasible, the priority number is increased by 1 until the first feasible move is detected which is 4. The priority 4 corresponds to the 12th disk in Fig.5.



Chromosome 1					
1	2	3	4	5	6
17	xx	16	15	14	7
18	xx	xx	xx	13	8
19	20	21	22	12	9
26	25	24	23	11	10

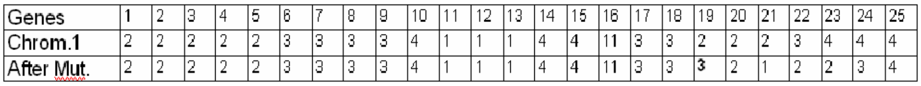
Chromosome 2					
1	22	21	23	24	25
2	xx	20	19	18	26
3	xx	xx	xx	17	16
4	7	8	11	12	15
5	6	9	10	13	14

Child Chromosome.1					
1	2	3	4	5	6
23	xx	22	21	20	7
24	xx	xx	xx	19	8
25	15	16	17	18	9
26	14	13	12	11	10

Fig. 5. Path generated based on the crossover of the sample chromosomes

Mutation Operator

The mutation operator is designed to incorporate the exploration impact. A randomly selected gene's value is changed from its current PP value to a feasible one arbitrarily. Following the mutated gene, all the moves again are generated with respect to its current PP. Although one gene change seems to be a modest change, it might result in a drastic change in the entire chromosome depending on the order of the gene selected for mutation. Due to this fact, we allowed only one mutation per chromosome. In Fig.6, the value of the gene 19 is changed from 2 to 3 arbitrarily. As a result, two-third of the remaining genes was changed yielding a different pattern.



Genes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Chrom.1	2	2	2	2	2	3	3	3	3	4	1	1	1	4	4	11	3	3	2	2	2	3	4	4	4
After Mut.	2	2	2	2	2	3	3	3	3	4	1	1	1	4	4	11	3	3	3	2	1	2	2	3	4

Fig. 6. A Illustration of the mutation operator (the number of the gene mutated=19)

The proposed pattern-based GA's typical parameters are set as follows: Population size: 500, maximum number of generations: 100, mutation probability:1/#ofGenes, crossover probability: 10%; tournament size: 2, crossover masking probability:70%, elite rate: 1%. In the proposed GA, the chromosomes of the current population, populations after crossover and after mutation are put together to build the enlarged population. Then selection operator reduces the size of the enlarged population to the original size again.

3 Experimental Results

The algorithm has been tested both in simulation environment, and at AI& R lab [12] test platform using Pioneer-3 DX mobile robots. A picture of the 840cmx720cm platform is given in Fig. 7.a. The map of the environment modeled with 116 disks is given in Fig. 7.b. The obstacles are represented with gray disks. It has three rooms and a hallway to connect the rooms. The environment is designed to pose a challenge for MRCPP methods. In Fig. 7.b, thick lines show constructed path by the proposed method. The total distance of movement of the proposed method is 115 units, and there is no repeated coverage. During the tests several additional obstacles are also placed into different parts of this environment to increase the complexity of the environment.

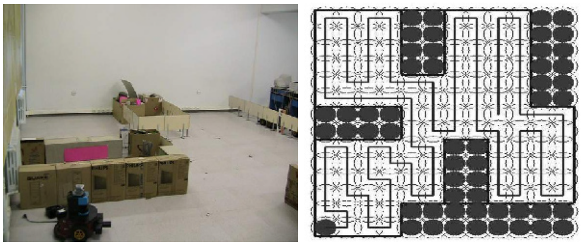


Fig. 7. AI&Robotic laboratory, **a)** Picture of the platform **b)** Map of the platform with a tour

In the following, the results of the proposed method are compared with the results of an existing grid based method, BSA [4]. BSA was chosen for comparison, because it divides the environment into individual cells and uses each cell as a decision unit to plan the coverage path similar to the approach presented in this paper. Another candidate method for comparison would be STC, but classical STC uses squares with dimension of four times of the robot's tool size to model the environment and assumes that if there is one obstacle in one of the squares, all four squares are treated as occupied. Therefore, some of free disks left as uncovered. Since our method aims complete coverage, it is not compared with STC. Later, experiments are carried out using Pioneer-3 DX robot in the laboratory environment.

3.1 Comparison of the Proposed Method with BSA

BSA is based on the execution of spiral filling paths. Before starting a spiral path, the robot is located nearby an obstacle which is situated in the reference lateral side RLS. RLS indicates the relative direction where obstacles are to be referenced during the spiral filling procedure. RLS is fixed in advance and can't be modified. OLS is the opposite lateral side, it identifies the antipode of RLS. The following set of four simple reactive rules allows the correct execution of the spiral coverage procedure [4]:

```
RS1 IF (obstacles_all_around): THEN ending_spiral_point_detected
RS2 IF (NOT obstacle_in_RLS): THEN turn_to(RLS) and move_forward
RS3 IF (obstacle_in_front): THEN turn_to(OLS)
RS4 OTHERWISE move_forward
```

Once the robot has reached the central ending point of a spiral filling path, the backtracking mechanism is invoked. It is employed to get back to unvisited areas, where a new spiral filling procedure can be performed. Backtracking points are detected and stored during the execution of a normal spiral path.

In order to compare the proposed method and the BSA, several obstacle configurations are used. In Figure 8.a-b, a five-disk obstacle is placed in the hallway blocking entrance of one room. Thick lines show the coverage path of the robot while the thick dashed-dot lines show redundant moves of the robot to access uncovered disks. The total distance of movement of the proposed method in Fig. 8.a is 114 units. For the same environment, the result of the BSA is 127 units as in Fig. 8.b. Therefore, 4 and 17 disks are covered repeatedly with the proposed method and BSA, respectively.

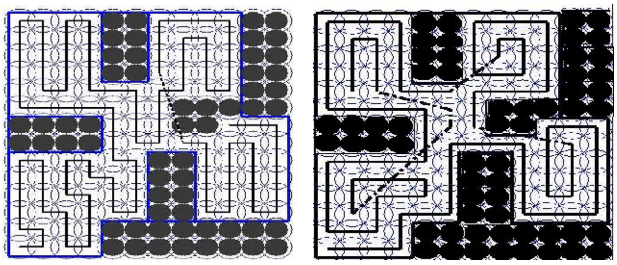


Fig. 8. a) Result of proposed method, **b)** Result of BSA

Due to page restrictions only one obstacle layout is given as in Fig. 8.a-b, and the results of the methods for five different obstacle configurations are summarized in Table 1. For each obstacle layout, ND.PM. denotes the number of disks for the proposed method, N.RD P.M. denotes the number of repeated disks for the proposed method, RC denotes percentage of repeated coverage, ND.PM. denotes the number of disks for the BSA, N.RD BSA. denotes the number of repeated disks for the BSA, and IMP denotes percentage improvement in the tour by the proposed method compared to BSA.

Table 1. Comparison the results of the proposed method and BSA

	N.D P.M.	N.RD P.M.	RC % P.M.	N.D. BSA	N.RD BSA	RC % BSA	IMP.%
Env#1	115	0	0	137	22	16	16
Env#2	113	0	0	137	24	17.5	17.5
Env#3	117	4	3.4	132	19	14.5	11.4
Env#4	112	1	1	132	20	15	15
Env#5	114	4	3.5	127	17	13.5	10

Env#1 denotes the environment in Fig.7 without obstacle, Env#2 denotes a two-disk obstacle in the middle, Env#3 denotes two one-disk obstacles placed at the room entrances, Env#4 denotes four-disks in bottom-left room, and Env#5 denotes five-disk obstacle blocking hallway as in Fig. 8. Figures of the environments #1-#4 could be reached from the web site [12]. Table 1 show that the proposed method has less repeatedly covered disk compared to BSA. Therefore, the robot may finish the coverage task in a shorter time and consuming less energy.

3.2 Experiments Using Pioneer 3-DX Mobile Robot

The proposed algorithm is applied to sensor-based coverage of an indoor environment using Pioneer 3-DX robot. The robot has an onboard P3-800 computer with Linux OS. The sensors on the robot are: SICK LMS 200 laser range finder, sonar ring sensors, camera, and compass. Aria ARLN software module is used for the localization purpose [13]. SICK LMS 200 laser range sensor is used for the sensor based coverage task. The sensor has normally a range of 50 meters, but for experimental purposes the range is restricted to 40cm with software.

An additional four-disk size obstacle is placed in the middle of the layout (Fig.7.b) to create a more challenging environment as in Fig. 9.a. The path of the proposed method is given in the Fig. 9.a is followed by the robot as given in Fig. 9.b. This figure is drawn using log values of the robot during the experiment. Fig. 9.c shows snapshot of the recorded video during the movement of the robot. The full video of the live performance of the robot is recorded, and can be reached from following web site: www.ai-robotlab.ogu.edu.tr [12].

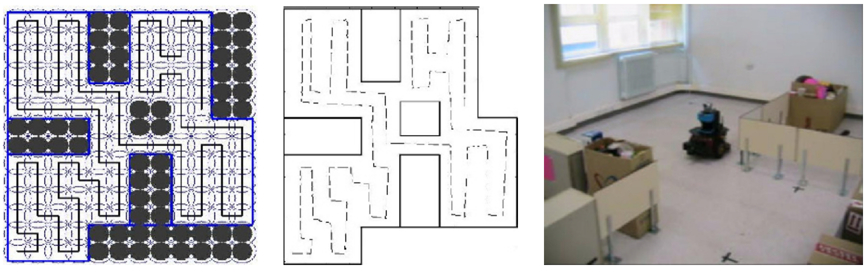


Fig. 9. a) Planned path using the proposed methods, b) The path followed by Pioneer-3 DX mobile robot in the laboratory, c) Snapshot of the recorded video

4 Conclusions

Sensor based coverage path planning for a mobile robot is one of the recent research issues. Among several approaches proposed, a few articles have addressed the use of the genetic algorithms for MRCPP problem. This article proposes prioritized movement patterns to generate efficient rectilinear coverage paths. They enable GA to find optimal or near-optimal solutions. Comparison with one of the existing methods in the literature, the proposed method resulted in 10% to 17.5% improvement in traveled distance over BSA for a set of challenging test environments with less repeated coverage. The results we obtained so far have been rather promising and efficient. The approach is yet to be studied for multiple robots cases.

Acknowledgment

This work is supported by the Scientific and Technical Research Council of Turkey (TUBITAK), Contract No 107E064, project title: "Mobile Robot Route Planning for Complete Coverage of Dynamic Indoor Environments".

References

1. Choset, H.: Coverage for robotics - A survey of recent results. Annals of Mathematics and Artificial Intelligence 31, 113–126 (2001)
2. Guo, Y., Balakrishnan, M.: Complete coverage control for nonholonomic mobile robots in dynamic environments. In: Proceedings of 2006 IEEE International Conference on Robotics and Automation, pp. 1704–1709 (2006)
3. Acar, E.U., Choset, H., Lee, J.Y.: Sensor-based coverage with extended range detectors. IEEE Transactions on Robotics 22(1), 189–198 (2006)
4. Gonzalez, E., Alvarez, O., Diaz, Y., Parra, C., Bustacara, C.: BSA: A Complete Coverage Algorithm. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 2040–2044 (2005)
5. Gabriely, Y., Rimon, E.: Spanning-tree based coverage of continuous areas by a mobile robot. Annals of Mathematics and Artificial Intelligence 31, 77–98 (2001)
6. Ryerson, A.E.F., Zhang, Q.: Vehicle Path Planning for Complete Field Coverage Using Genetic Algorithms. Agricultural Engineering International: the CIGR E journal IX, 1–11 (2007) (Manuscript, ATOE-07-014)
7. Jimenez, P.A., Shirinzadeh, B., Nicholson, A., Alici, G.: Optimal area covering using genetic algorithms. In: Proceedings of the 2007 IEEE /ASME International Conference on Advanced Intelligent Mechatronics, pp. 1–5 (2007)
8. Kapanoglu, M., Ozkan, M., Yazici, A., Parlaktuna, O.: A Genetic Algorithm with Orientation Compasses for Single or Multi-Robot Coverage Path Planning. In: Proceedings of 6th International Symposium on Intelligent & Manufacturing Systems, pp. 668–678 (2008)
9. Goldberg, D.E.: Genetic Algorithm in search, optimization, and Machine Learning. Addison-Wesley, Reading (1989)
10. Castillo, O., Trujillo, L., Melin, P.: Multiple objective genetic algorithms for path-planning optimization in autonomous mobile robots. Soft Computing 11(3), 269–279 (2007)
11. Erinc, G., Caprin, S.: A genetic algorithm for nonholonomic motion planning. In: Proceedings of the 2007 IEEE International Conference on Robotics and Automation, pp. 1843–1849 (2007)
12. Artificial Intelligence and Robotic laboratory, Eskisehir Osmangazi University, Turkey, `http://www.ai-robotlab.ogu.edu.tr`
13. Aria- ARNL software module (2009), `http://robots.mobilerobots.com`

Economic Models with Chaotic Money Exchange

Carmen Pellicer-Lostao and Ricardo López-Ruiz

Department of Computer Science and BIFI,
Universidad de Zaragoza, 50009 - Zaragoza, Spain
carmen.pellicer@unizar.es, rilopez@unizar.es

Abstract. This paper presents a novel study on gas-like models for economic systems. The interacting agents and the amount of exchanged money at each trade are selected with different levels of randomness, from a purely random way to a more chaotic one. Depending on the interaction rules, these statistical models can present different asymptotic distributions of money in a community of individuals with a closed economy.

Keywords: Complex Systems, Econophysics, Gas-like Models, Money Dynamics, Random and Chaotic numbers, Modeling and Simulation.

1 Introduction

Econophysics is born as a new science devoted to study Economy and Financial Markets with the traditional methods of Statistical Physics [1]. This discipline applies many-body techniques developed in statistical mechanics to the understanding of self-organizing economic systems [2]. One of its main objectives is to provide economists with new tools and new insights to deal with the complexity arising in economic systems.

The seminal contributions in this field [3], [4], [5] have to do with agent-based modeling and simulation. In these works, an ensemble of economic agents in a closed economy is interpreted as a gas of interacting particles exchanging money instead of energy. Despite randomness is an essential ingredient of these models, they can reproduce the asymptotic distributions of money, wealth or income found in real economic systems [2].

In the work presented here, the transfers between agents are not completely random as in the traditional gas-like models. The authors introduce some degree of determinism, and study its influence on the asymptotic wealth distribution in the ensemble of interacting individuals. As reality seems to be not purely random [6], the rules of agent selection and money transfers are selected with different levels of randomness and extended up to chaotic conditions. This unveils their influence in the final wealth distribution in diverse ways. This study records the asymptotic wealth distributions displayed by all these scenarios of simulation.

The paper is organized as follows: Section 2 introduces the basic theory of gas-like economic models. Section 3 describes the four simulation scenarios studied in this work, and the following sections show the results obtained in the simulations. Conclusions are discussed in the final section.

2 The Gas-Like Model: Boltzmann-Gibbs Distribution of Money

The conjecture of a kinetic theory of (ideal) gas-like model for trading in market was first discussed in 1995 [7]. Then, it was in year 2000, when several noteworthy papers dealing with the distribution of money and wealth [3], [4], [5] presented this theory in more detail.

The gas-like model for the distribution of money assimilates the dynamics of a perfect gas, where particles exchange energy at every collision, with the dynamics of an economic community, where individuals exchange money at every trade. When both systems are closed and the magnitude of exchange is conserved, the expected equilibrium distribution of these statistical systems may be the exponential Boltzmann-Gibbs distribution.

$$P(x) = ae^{-x/b} \tag{1}$$

Here, a and b are constants related to the mean energy or money in the system, $a =< x >^{-1}$ and $b =< x >$. Theoretically, the derivation (and so the significance) of this distribution is based on the statistical behavior of the system and on the conservation of the total magnitude of exchange. It can be obtained from a maximum entropy condition [8] or from purely geometric considerations on the equiprobability over all accessible states of the system [9].

Different agent-based computer models of money transfer presenting an asymptotic exponential wealth distribution can be found in the literature [3], [10] , [11]. In these simulations, a community of N agents with an initial quantity of money per agent, m_0, trade among them. The system is closed, hence the total amount of money M is a constant ($M = N * m_0$). Then, a pair of agents is selected (i, j) and a bit of money Δm is transferred from one to the other. This process of exchange is repeated many times until statistical equilibrium is reached and the final asymptotic distribution of money is obtained.

In these models, the rule of agents selection in each transaction is chosen to be random (no local preference or no intelligent agents). The money exchange Δm at each time is basically considered under two possibilities : as a fixed or as a random quantity. From an economic point of view, this means that agents are trading products at a fixed price or that prices (or products) can vary freely, respectively.

These models have in common that they generate a final stationary distribution that is well fitted by the exponential function. Perhaps one would be tempted to affirm that this final distribution is universal despite the different rules for the money exchange, but this is not the case as it can be seen in [3], [10].

3 Simulation Scenarios

Real economic transactions are driven by some specific interest (or profit) between the different interacting parts. Thus, on one hand, markets are not purely random. On the other hand, the everyday life shows us the unpredictable component of real economy. Hence, we can sustain that the short-time dynamics of economic systems evolves under deterministic forces and, in the long term, the recurrent crisis happening in these kind of systems show us the inherent instability of them. Therefore, the prediction of the future situation of an economic system resembles somehow to the weather prediction. We can conclude that determinism and unpredictability, the two essential components of chaotic systems, take part in the evolution of economy and financial markets.

Taking into account these evidences, the study of gas-like economic models where money exchange can have some chaotic ingredient is an interesting possibility. In other words, one could consider an scenario where the selection rules of agents and regulation of products prices in the market are less random and more chaotic. Specifically, this paper considers the selection of interacting values with different levels of randomness up to chaotic regime.

In the computer simulations presented here, a community of N agents is given with an initial equal quantity of money, m_0, for each agent. The total amount of money, $M = N * m_0$, is conserved in time. For each transaction, a pair of agents (i, j) is selected, and an amount of money Δm is transferred from one to the other. In this work two simple and well known rules are used. Both consider a variable v in the interval $(0, 1)$, not necessarily random, in the following way:

- **Rule 1**: the agents undergo an exchange of money, in a way that agent i ends up with a v-dependent portion of the total of two agents money, $(v * (m_i + m_j))$, and agent j takes the rest $((1 - v) * (m_i + m_j))$ [10].
- **Rule 2**: an v-dependent portion of the average amount of the two agents money, $\Delta m = v * (m_i + m_j)/2$, is taken from i and given to j [3]. If i doesn't have enough money, the transfer doesn't take place.

As there are two different simulation parameters involved in these gas-like models (the parameter for selecting the agents involved in the exchanges and the parameter defining the economic transactions), four different scenarios can be obtained depending on the random-like or chaotic election of these parameters. These scenarios are considered in the following sections and are described as:

- **Scenario I**: random selection of agents with random money exchanges.
- **Scenario II**: random selection of agents with chaotic money exchanges.
- **Scenario III**: chaotic selection of agents with random money exchanges.
- **Scenario IV**: chaotic selection of agents with chaotic money exchanges.

It is worthy to say at this point, that the words *random* and *pseudo-random* are used in the same sense, where random refers to the stochastic character of the process and pseudo-random refers to the computational simulation used to produce the process. The pseudo-random and chaotic numbers are obtained with two chaotic pseudo-random bit generators, selected to this purpose. Technical details of these generators are fully described in [12] and [13], and are based in two 2D chaotic systems: the Hénon Map and the Logistic Bimap. Interactive animations of them can also be seen in [14].

The particular properties of these generators [12], [13] make them suitable for the purpose of this study. They are able to produce pseudo-random and chaotic patterns of numbers that can be used as parameters of the simulations. Basically, these generators have two parts: the output of the chaotic maps is used as input of a binary mixing block that randomizes the chaotic signals and generates the final random numbers. Then, on one side, it is possible to take the exit of the chaotic blocks and produce chaotic sequences of numbers. On the other side, they can generate a sequence of numbers with a gradual variation of randomness by controlling a delay parameter P in the binary mixing block.

This last feature is obtained by varying the *shift factor* P ($P > 1$) in a way that the lower its value, the worse is the random quality of the numbers generated. Specifically, there is also a P_{min} (around 80) above which, the properties of the generator can be considered of good random quality.

As an example to show this gradual variation of randomness, the generator in [13] is used to produce different binary sequences and initial conditions of $S2$ (further details [13]). These bits are transformed in integers of 32 bits and transformed to floats dividing by the constant $MAXINT = 4294967295$.

When the shift factor P is varied, the random quality of these binary sequences also varies. This can be statistically measured by submitting them to statistical tests and it can be also graphically observed in Fig.1.

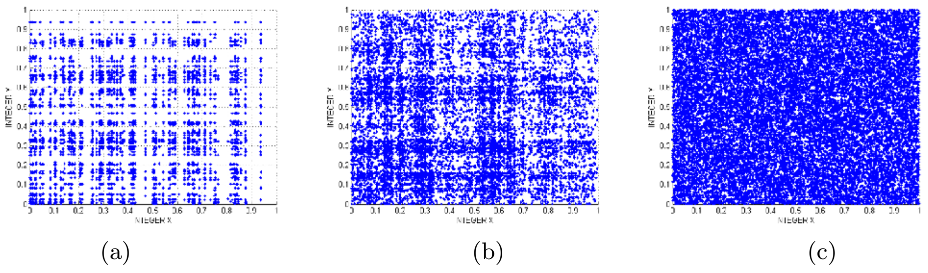


Fig. 1. Representation of 20000 integers as pairs of floats in the interval $[0, 1] \times [0, 1]$. The quality of their randomness improves as the shift factor P grows in magnitude. (a) $P = 1$ (b) $P = 5$ (c) $P = 110$.

In Fig.1, we generated 640000 bits to obtain 20000 integers of 32 bits. The integers obtained from the generator are transformed to floats. The variation of the shift factor shows graphically that with no shift at all, $P = 1$, the integers obtained are hardly random. With $P = 5$, the bits generated do not pass the frequency or monobit test and still show a strong no random appearance. When the shift factor grows over $P = 110$, the binary sequences pass Diehard and NIST statistical tests. Graphically in Fig.1(d), it can be assessed to possess high random quality.

4 Scenario I: Random Selection of Agents with Random Money Exchange

In this section, both simulation parameters are selected to obey certain pseudo-random patterns. Thus, the generators in [13] is used to produce different binary sequences (with initial conditions of $S2$ see further details in [13]). These bits are transformed in integers of 32 bits and used as simulation parameters to select the agents or the money to exchange.

Then, computer simulations are performed in the following manner. A community of $N = 500$ agents is considered with an initial quantity of money of $m_0 = 1000\$$. For each transaction two integer numbers are selected from the generated pseudo-random sequence with a given shift factor $P = P_{Ag}$. A pair of agents (i, j) is selected according to these integers with an N-modulus operation. Additionally, a third integer number is obtained from another pseudo-random sequence with another shift factor $P = P_{Ex}$. This integer is used to obtain a float number v in the interval $[0, 1]$. The value of v and the rule selected (Rule 1 or Rule 2) for the exchange determine the amount of money Δm that is transferred from one agent to the other.

Choosing $P = P_{Ag}$ with different values it is possible to emulate an environment where the agents are locally selected under a more or less random scenario situation. The same for $P = P_{Ex}$, the prices of products or services in the market can be emulated to be less random, regulated, or completely arbitrary.

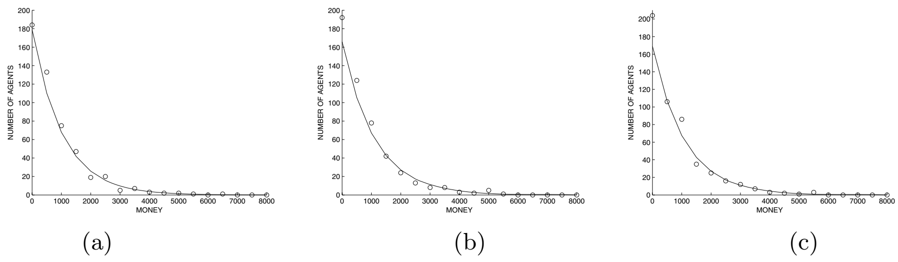


Fig. 2. Simulation of Scenario I. (a) $P_{Ag} = 2$ and $P_{Ex} = 110$ for Rule 1, (b) $P_{Ag} = 110$ and $P_{Ex} = 3$ for Rule 2 and (c) $P_{Ag} = 2$ and $P_{Ex} = 5$ for Rule 2.

The simulations take a total time of 400000 transactions. Two rules of money exchange were considered, Rule 1 and 2 described in Section 2. The results show that all cases, independently of the value of the shift factor, produce a stationary distribution that is well fitted to the exponential function. In Fig. 2, some particular cases are shown, taking pseudo-random selection of agents or pseudo-random calculation of Δm. Although not depicted in Fig. 2 the case, where both agents and traded money are selected randomly, gives very similar results to the cases shown here and also similar to the ones obtained in [3].

5 Scenario II: Random Selection of Agents with Chaotic Money Exchange

In the previous section, it is observed that a variation in the random degree of selection of agents and/or traded money, does not affect the final equilibrium distribution of money. It leads to an exponential in all cases. In this section, the selection of agents is going to be set to random, while the exchange of money is going to be forced to evolve according to chaotic patterns. Economically, this means that the exchange of money has a deterministic component, although it varies chaotically. Put it in another way, the prices of products and services are not completely random. On the other side, the interaction between agents is arbitrary, as they are randomly chosen.

Let us take the chaotic pseudo-random generators a step backwards, directly at the output of the chaotic block with initial conditions $S2$ and $R1$ (see [13] and [12] respectively, for details). Now the chaotic map variables x_i and y_i can be used as simulation parameters. Consequently, the computer simulations are performed in the following manner. A community of $N = 500$ agents is considered with an initial quantity of money of $m_0 = 1000\$$. For each transaction two random numbers from a standard random generator are used to select a pair of agents. Additionally, a chaotic float number is produced to obtain the float number υ in the interval $[0, 1]$. The value of υ is calculated as $|x_i|/1.5$ for the Hénon map and as x_i for the Logistic Bimap. This value and the rule selected for the exchange determine the amount of money Δm that is transferred from one agent to the other.

The simulations take a total time of 400000 transactions. Different cases are considered, taking the Hénon chaotic map or the Logistic Bimap. Rules 1 and 2 are also considered. New features appear in this scenario. These can be observed in Fig.3.

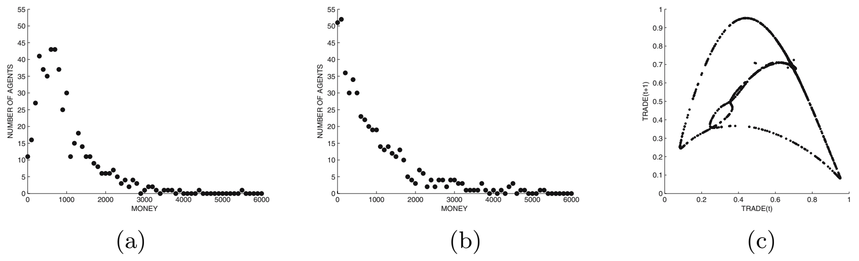


Fig. 3. Simulation of Scenario II where agents are selected randomly and the money exchange follows chaotic patterns. (a) Chaotic trade selection with Logistic Bimap and Rule 1, (b) Chaotic trade selection with Logistic Bimap and Rule 2 and (c) Chaotic trade points from the Logistic Bimap used in the simulations and represented as pairs in the range $[0, 1] \times [0, 1]$.

The first feature is that the chaotic behavior of υ (see Fig.3 (c)) is producing a different final distribution for each rule. Rule 2 is still displaying the exponential shape, but Rule 1 gives a different asymptotic function distribution. It presents a very low proportion of the population in the state of poorness, and a high percentage of it in the middle of the richness scale, near to the value of the mean wealth. Rule 1 seems to lead to a more equitable distribution of wealth.

Basically, this is due to the fact that Rule 2 is asymmetric. Each transaction of Rule 2 represents an agent i trying to buy a product to agent j and consequently agent i always ends with the same o less money. On the contrary, Rule 1 is symmetric and in each interaction both agents (i, j), as in a joint venture, end up with a division of their total wealth. Now, think in the following situation: with a fixed υ, let say $\upsilon = 0.5$, Rule 1 will end up with all agents having the same money as in the beginning, $m_0 = 1000\$$. Using a chaotic evolution of υ means restricting its value to a defined region, that of the chaotic attractor. Consequently, this is enlarging the distribution around the initial value of 1000\$ but it does not go to the exponential as in the random case [10].

6 Scenario III: Chaotic Selection of Agents with Random Money Exchange

In this section, the selection of agents is going to evolve chaotically, while the exchange of money is random. Economically, this means that the locality of the agents or their preferences to exchange are somehow deterministic but under complex evolution. Thus, some commercial relations are going to be restricted. On the other hand, regulation of prices is random and they are evolving freely.

The chaotic generators are used directly at the output of the chaotic block, exactly as in the previous section. Again a community of $N = 500$ agents with initial money of $m_0 = 1000\$$ is taken and the chaotic map variables x_i and y_i will be used as simulation parameters. For each transaction two chaotic floats in the interval $[0, 1]$ are produced. The value of these floats are $|x_i|/1.5$ and $|y_i|/0.4$ for the Hénon map and x_i and y_i for the Logistic Bimap. These values are used to obtain i and j as in previous section. Additionally, a random number from a standard random generator are used to obtain the float number υ in the interval $[0, 1]$. The value of υ and the selected rule determine the amount of money Δm that is transferred from one agent to the other.

The simulations take a total time of 400000 transactions. Different cases are considered, taking the Hénon chaotic map or the Logistic Bimap, and Rules 1 and 2. As a result, an interesting point appears in this scenario with both rules. This is the high number of agents that keep their initial money in Fig. 4(a) and (b). The reason is that they don't exchange money at all. The chaotic numbers used to choose the interacting agents are forcing trades between a deterministic group of them and hence some commercial relations result restricted.

In can be observed in Fig. 4(a) and (b), that the asymptotic distributions in this scenario again resemble the exponential function. The Logistic Bimap is symmetric (coordinates x_i and y_i) and it produces the effect of behaving like scenario II but with a restricted number of agents.

Amazingly, the Hénon Map with Rule 2 in Fig. 4(c) leads to a distribution with a heavy tail, a Pareto like distribution. A high proportion of the population (around 420 agents) finish in the state of poorness. Though not completely shown in this figure, there are a minority of agents with great fortunes distributed up to the range of 40000\$. This is due to the asymmetry of the rule, where agent

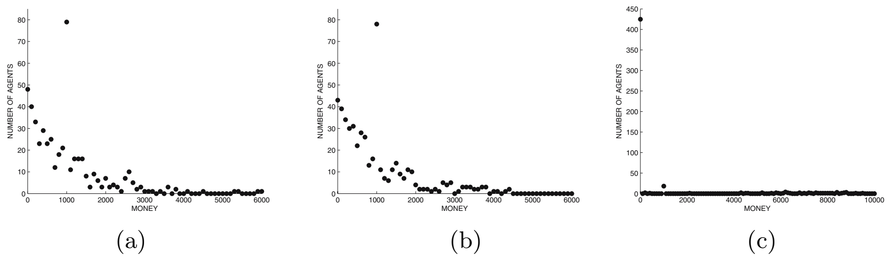


Fig. 4. Simulation of Scenario III where agents are selected chaotically and the money exchange is set to be random. (a) Chaotic agents selection with Logistic Bimap and Rule 1, (b) Chaotic agents selection with Logistic Bimap and Rule 2, (c) Heavy tail distribution of chaotic agents selection with Hénon map and Rule 2.

i always decrements its money, and the asymmetry of coordinates x_i and y_i in the Hénon chaotic Map used for the selection of agents. This double asymmetry makes some agents prone to loose in the majority of the transactions, while a few others always win.

7 Scenario IV: Chaotic Selection of Agents with Chaotic Money Exchange

In this section, the selection of agents and the exchange of money are chaotic. Economically, this means that commercial relations are complex and some transactions are restricted. The money exchange varies disorderly, but in a more deterministic way. The prices of products and services are not completely random.

As in the previous sections, the chaotic generators are used directly at the output of the chaotic block. Again the chaotic map variables, x_i and y_i, will be used as simulation parameters. The computer simulations are performed in the following manner. A community of $N = 500$ agents is considered with an initial quantity of money of $m_0 = 1000\$$. For each transaction, four chaotic floats in the interval $[0, 1]$ are produced. Two of these floats are $|x_i|/1.5$ and $|x_{i+1}|/1.5$ for the Hénon map or x_i and y_i for the Logistic Bimap. These values are used to obtain i and j through simple multiplication (i.e.: $i = x_i * N + 1$). Additionally, a chaotic float number is produced to obtain the float number v in the interval $[0, 1]$. The value of v is calculated as $(|y_i| + |y_{i+1}|)/0.8$ for the Hénon map or as $(x_{i+1} + y_{i+1})/2$ for the Logistic Bimap. This value and the selected rule of exchange determine the money Δm that is transferred between agents.

The simulations take a total time of 400000 transactions. Also, in this scenario, we take the Hénon chaotic map or the Logistic Bimap, and Rules 1 and 2 are considered. As a result, the same properties of both asymptotic distributions are maintained respect to section 5, then the same differences between rules are observed, as shown in Fig.5

Here, again a high number of agents keep their initial money. The chaotic choice in Fig.5 (c) is forcing trades between a specific group of agents, and then

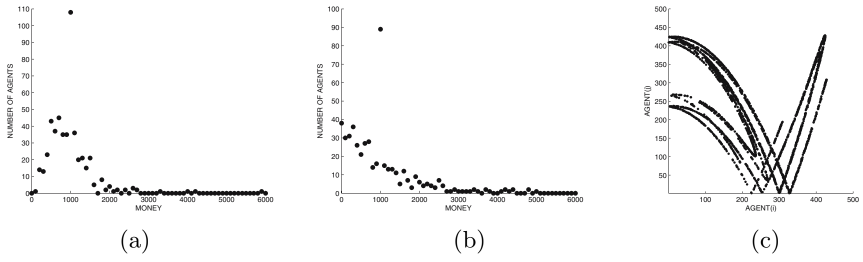


Fig. 5. Simulation of Scenario IV where agents and money exchange are chaotic. (a) Chaotic selection with Henon Map and Rule 1, (b) Chaotic selection with Henon Map and Rule 2 and (c) Chaotic agent points used in these simulations represented as pairs (i,j) in the range $[1,500] \times [1,500]$.

this type of locality makes some commercial relations restricted. The different behavior for Rule 1 and 2 is similar to scenario III. Rule 2 still presents an exponential shape, but Rule 1 gives a different function distribution with a maximum near the mean wealth. We remark that Rule 1 is able to generate a more equitable society when chaotic mechanisms are implemented in both processes, the agents selection and the money transfer.

8 Conclusions

The work presented here focuses on the statistical distribution of money in a community of individuals with a closed economy, where agents exchange their money under certain evolution laws. The several theoretical models and practical simulation results in this field, implement rules where the interacting agents or the money exchange between them are traditionally selected as fixed or random parameters ([2], [4], [10]).

Here, a novel perspective is introduced. As reality tends to be more complex than purely fixed or random, it seems interesting to consider chaotic driving forces in the evolution of the economic community. Therefore, a series of agent-based computational results has been presented, where the parameters of the simulations are selected with different levels of randomness and extended up to chaotic conditions.

In a first scenario, the exponential Boltzmann-Gibbs distribution is obtained for two different rules of money exchange under conditions with different levels of randomness. Consequently, this does not distinguish between different evolution rules and richness is shared among agents in an exponential and unequal mode.

Introducing chaotic parameters in three other different scenarios leads to different results, in the sense that restriction of commercial relations is observed, as well as a different asymptotic wealth distribution depending on the rule of money exchange. It is remarkable that a more equitable distribution of wealth is obtained in one of the evolution rules when some of the dynamical parameters are driven by a chaotic system. This can be qualitatively observed in the distributions of money that have been obtained and reported for two different scenarios.

The authors hope that this study can trigger other works that continue to provide new clues in the nature of economic self-organizing systems.

Acknowledgements. The authors acknowledge some financial support by Spanish grant DGICYT-FIS200612781-C02-01.

References

1. Mantegna, R., Stanley, H.E.: An Introduction to Econphysics: Correlations and Complexity in Finance. Cambridge University Press, Cambridge (1999)
2. Yakovenko, V.M.: Econophysics, Statistical Mechanics Approach to. To appear in Encyclopedia of Complexity and System Science. Springer, Heidelberg (to appear, 2009) arXiv:0709.3662v4 [q-fin.ST]
3. Dragulescu, A.A., Yakovneko, V.M.: Statistical mechanics of money. The European Physical Journal B 17, 723–729 (2000)
4. Chacraborti, A., Chacrabarti, B.K.: Statistical mechanics of money: how saving propensity affects its distribution. The European Physical Journal B 17, 167–170 (2000)
5. Bouchaud, J.P., Mézard, M.: Wealth condensation in a simple model economy. Physica A 282, 536–545 (2000)
6. Sanchez, J.R., Gonzalez-Estevez, J., Lopez-Ruiz, R., Cosenza, M.: A Model of Coupled Maps for Economic Dynamics. European Physical Journal Special Topics 143, 241–243 (2007)
7. Chakrabarti, B.K., Marjit, S.: Self-organization in Game of Life and Economics. Indian Journal Physics B 69, 681–698 (1995)
8. Jaynes, J.T.: Information Theory and Statistical Mechanics. Physical Review E 106, 620–630 (1957)
9. López-Ruiz, R., Sañudo, J., Calbet, X.: Geometrical derivation of the Boltzmann factor. American Journal of Physics 76, 780–781 (2008)
10. Patriarca, M., Chakraborti, A., Kaski, K.: Gibbs versus non-Gibbs distributions in money dynamics. Physica A 340, 334–339 (2004) arXiv:cond-mat/0312167v1
11. Hayes, B.: Follow the money. American Scientist 90, 400 (2002)
12. Suneel, M.: Cryptographic Pseudo-Random Sequences from the Chaotic Hénon Map (2006) arXiv:cs/0604018v2 [cs.CR]
13. Pellicer-Lostao, C., López-Ruiz, R.: Pseudo-Random Bit Generation based on 2D chaotic maps of logistic type and its Applications in Chaotic Cryptography. In: Gervasi, O., Murgante, B., Laganà, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2008, Part II. LNCS, vol. 5073, pp. 784–796. Springer, Heidelberg (2008)
14. Pellicer-Lostao, C., López-Ruiz, R.: Orbit Diagram of the Hénon Map and Orbit Diagram of Two Coupled Logistic Maps from The Wolfram Demonstrations Project, http://demonstrations.wolfram.com/

Knowledge Aware Bisimulation and Anonymity*

Han Zhu[1], Yonggen Gu[2], and Xiaojuan Cai[1]

[1] Basics Lab, Department of Computer Science
Shanghai Jiao Tong University, Shanghai, China
{zhu-h,cxj}@sjtu.edu.cn

[2] Department of Computer Science and Technology
Huzhou Teachers College, Huzhou, China
gyg68@hutc.zj.cn

Abstract. We propose a knowledge aware bisimulation equivalence relation on the Calculus of Applied Pi Calculus. Applied Pi is well-known for discribing and analyzing security protocols. Our equivalence relation is especially useful in analyzing the property of anonymity. We give an analysis of iKP anonymity as a running example to show the effectiveness of this approach.

Keywords: Network Security, Formal Methods, Applied Pi, Anonymity.

1 Introduction

The Calculus of Applied Pi, pioneered by Abadi and Fournet [1] is an extension of π-calculus [2,3]. In the field of analyzing security protocols, where knowledge and term equations play an important rule in guaranteeing security properties of protocols (secrecy, authentication, etc), Applied Pi is the most suitable process calculus. It has a more general setting of terms to model most security primitives nowadays, and this setting provides a mechanism to model unknown new primitives which would appear in future. The calculus itself is decoupled from terms, so its algebraic property is not affected by which primitives its terms are modeling. It has been used to model Just Fast Keying [4], certified email [5], private authentication [6], etc.

In Applied Pi, the security properties such as secrecy and authentication are captured by the labeled bisimilarity [1], which is defined upon labeled transition system. However labeled bisimilarity does not seem to be a suitable equivalence relation between processes for security protocols. One reason is that it does not take account of knowledge. Secondly it is not a congruent relation. It is also coarser than we want. For example, given the equational theory

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x$$

* This work is supported by the National 973 Project (2003CB317005), the National Nature Science Foundation of China (60573002, 60703033), the Nature Science Foundation of Huzhou City Zhejiang Province (2007YZ10) and the Sci-Tech Brainstorm Stress Project of Huzhou City Zhejiang Province (2007G22).

let us take the following two processes:

$$\begin{aligned} A \equiv\ & \nu m.(\bar{a}\langle \mathsf{enc}(m,k)\rangle.b(x).(\tau.\bar{c}\langle c\rangle + \tau.\bar{d}\langle d\rangle \\ & + \tau.\mathsf{if}\ \mathsf{dec}(x,k) = m\ \mathsf{then}\ \bar{c}\langle c\rangle\ \mathsf{else}\ \bar{d}\langle d\rangle)) \\ B \equiv\ & \nu m.(\bar{a}\langle \mathsf{enc}(m,k)\rangle.b(x).(\tau.\bar{c}\langle c\rangle + \tau.\bar{d}\langle d\rangle)) \end{aligned}$$

One can easily check that A is labeled bisimilar to B since

$$\mathsf{dec}(\mathsf{enc}(m,k),k) = k$$

is one of the equations in the equational theory. But from the viewpoint of an practical observer, these two processes should not be bisimilar. Because the third summand of A,

$$\tau.\mathsf{if}\ \mathsf{dec}(x,k) = m\ \mathsf{then}\ \bar{c}\langle c\rangle\ \mathsf{else}\ \bar{d}\langle d\rangle$$

could tell apart the content it receives on channel b by sending something on channel c or d. While process B does not have this ability.

The basic idea is straightforward. A knowledge based environment with time precedence relation is used to deal with the bounded names. The environment records the history of a process when it is running. The time precedence relation records those pairs of names and terms. Its role is to inhibit from substituting a later emitted term for a free substitutable name. The resulting equivalence relation is finer than labeled bisimilarity enough to rule out the unreasonable equality of A and B in the above example. It also enjoys the full congruent property, which requires an equivalent relation closed under every operator including input prefix. In the fields of analyzing security protocols, a congruent equivalence relation means we can verify security properties of a large system by break up to small parts, then each small part satisfies a security property implies the whole system also satisfies it.

Related Works. Using process calculi to analyze network security protocols is firstly studied by Lowe [7], wherein a flaw of Needham-Schroeder key distribution protocol is found by modeling the protocol in CSP and checking it with FDR. Later on, some *ad hoc* calculi are proposed, among which the Spi Calculus [8] and the Applied Pi Calculus [1] are the most widely studied. Many knowledge aware equivalence relations, like framed bisimulation [9], hedged bisimulation [10] to name a few, on Spi have been proposed to extend standard bisimulation equivalence which lacks knowledge representation.

Anonymity, also called untraceability, is firstly proposed by Chaum [11] to solve the Dinning Cryptographer Problem. Schneider and Sidiropoulos analyze anonymity with CSP [12]. In [13] Kremer and Ryan analyzed the FOO92 voting protocol in Applied Pi and prove that it satisfies anonymity. Chothia [14] uses bisimulation in the π-calculus to test the anonymity of an anonymous file-sharing system. Cai [15] uses Probabilistic Applied Pi to give anonymity a quantitative analysis. The anonymity of iKP is studied in [16]. Principals of the protocol are modeled as Applied Pi processes and the anonymity is captured by static equivalence. The work is instructive, however, their formulation of anonymity is unclear and the proof is not correct though the results do hold. We give an unambiguous formulation of anonymity based on the open bisimilarity and provide a sound proof of it.

Outline of the Paper. The rest of the paper is organized as follows. Section 2 briefly introduces the Applied Pi Calculus and develops knowledge aware bisimilarity and proves its properties. Improvements on analyzing iKP anonymity is shown in Section 3. Finally Section 4 concludes and points out some directions for future research. All technical details and additional information can be found in [17].

2 Applied Pi and Open Bisimulation

2.1 Applied Pi

In this subsection, we briefly recall the syntax and semantics of the Applied Pi Calculus. Readers are referred to [1] for full details.

We presume a countable set of names $\mathcal{N}_c$, ranged over by $a, b, \ldots$ and their decorated forms, and a countable set of variables $\mathcal{N}_v$, ranged over by $x, y, \ldots$ and their decorated forms. $\tilde{n}$ denotes some finite set of names $\{n_1, \ldots, n_k\}$. A signature Σ is the set of all function symbols needed to model some protocols.

Terms are defined as follows:

$$L, M, N, T ::= a, b, c, \ldots \mid x, y, z, \ldots \mid \mathsf{f}(M_1, \ldots, M_l)$$

They are constructed from names, variables and function application on terms. A *substitution* σ is a map from variables to terms. As in π-calculus, names are used to express channels, and set of channels is denoted by $Chan$. Names can represent atomic data such as keys, nonces, random numbers as well. Function applications on terms are proposed to model all kinds of cryptographic operations such as *encryption* and *decryption*. The relationship of cryptographic primitives is described by an equational theory E. For example, when we are dealing with symmetric cryptography, we could let $\Sigma = \{\mathsf{enc}, \mathsf{dec}\}$ and E has the following equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$.

Plain processes are constructed by commonly seen operators:

$$\begin{aligned} P, Q, R ::= \mathbf{0} \mid u(x).P \mid \bar{u}\langle N \rangle.P \mid P \,|\, Q \mid\, !P \\ \mid \nu n.P \mid \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \end{aligned}$$

Extended processes are plain processes paralleled by active substitutions and the restrict operator also has effect on active substitutions:

$$A, B, C ::= P \mid A \,|\, B \mid \nu n.A \mid \nu x.A \mid \{M/x\}$$

The differences between plain processes and extended processes are *active substitutions*. The notation $\{M/x\}$ is an active substitution which replaces the variable x with the term M. The active substitution $\{M/x\}$ typically appears when the term M has been sent out, and the environment could receive it. The *frame* of an extended process is the active substitution part of an extended process. It can be written in the form of $\phi_A = \nu\tilde{n}.\{\tilde{M}/\tilde{x}\}$ and $\tilde{n} \subseteq n(\tilde{M})$. Every extended process can be mapped into its frame by removing plain processes.

We write $fn(A)$ and $bn(A)$ for free and bound names of A. The set of names that occur in A is denoted as $n(A) = bn(A) \cup fn(A)$. If A is an extended process, the frame of A is the active substitution part of it. Function $dom(\phi)$ returns the domain of a frame ϕ. The set of frames are denoted by $Frame$.

Frames can be viewed as the static knowledge exposed by A to the environment, so there comes the *deduction* [18] which can be done by the environment. We write $\phi \vdash M$ to mean that M can be deduced from ϕ.

Definition 1. *We say that two terms M and N are* equal *in the frame ϕ, and write $(M = N)\phi$, if and only if $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma = N\sigma$ and $\{\tilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\tilde{n}$ and substitution σ.*

Definition 2. *We say that two closed frame ϕ and ψ are* statically equivalent, *denoted by $\phi \approx_s \psi$ if $dom(\phi) = dom(\psi)$ and for all terms M, N, we have $(M = N)\phi$ if and only if $(M = N)\psi$.*

Definition 3 (Static Equivalence $\approx_s$). *We call two closed extended processes A, B* statically equivalent, *and write $A \approx_s B$, if their frames are statically equivalent.*

Definition 4. *Labeled bisimilarity $\approx_l$ is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A \mathcal{R} B$ implies,*

1. *$A \approx_s B$;*
2. *if $A \xrightarrow{\tau} A'$, then $B \xrightarrow{\tau}^* B'$ and $A'\mathcal{R}B'$ for some B';*
3. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq dom(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \xrightarrow{\tau}^* \xrightarrow{\alpha} \xrightarrow{\tau}^* B'$ and $A' \mathcal{R} B'$ for some B'.*

The decidability of the equational theory is crucial in automatically checking equality between two Applied Pi processes. Some positive results have been reported in [18]. For example, deducibility is decidable in polynomial time for a large class of equational theories including the theories for encryption, decryption and digital signatures. In this paper, we assume that the deducibility of equational theories are always decidable.

2.2 Open Bisimulation

The active substitution is already a good candidate for the knowledge aware environment. However it will be absorbed by an input action. This causes some information lost after an interaction, so we reserve the time precedence constraint on substitutions. Since a term can substitute for a free name, our precedence relation is defined on $Frame \times (\mathcal{N}_c \cup \mathcal{N}_v)$. The frame of a process is obtained by mapping all plain processes to $\mathbf{0}$, and all active substitutions remained.

The environment will be the form of $e = \langle V, \prec \rangle$, plus the implicit frames of extended processes. V is the set of substitutable names and variables, and $\prec$ is time precedence relation. The constraint on substitution is achieved by the so-called respectiveness of a substitution to an environment $\sigma \rhd e$. As the bisimulation game is playing, the environment should be updated and extended accordingly.

In Applied Pi, it is not required that the same σ applies to both P and Q, since two key protected cyphertexts are thought to be equal when the key is unknown to observers. We use a tuple $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2 \rangle$, whose first part σ_1 applies to P and σ_2 applies to Q. From now on, we write bold face $\boldsymbol{A}$ to denote a tuple in the form of $\langle A_1, A_2 \rangle$.

Definition 5. *A tuple $\langle V, \prec \rangle$ is an* environment*, if $V \subseteq (\mathcal{N}_c \cup \mathcal{N}_v)$ is a finite set of substitutable names and variables from both two sides, $\prec = \langle \prec_1, \prec_2 \rangle$ and $\prec_i \subseteq Frame \times V (i \in \{1, 2\})$ is the precedence relation between frames and substitutable names and variables. The set of all environments is denoted by E.*

The intuition behind an environment $e = \langle V, \prec \rangle$ is to make a clear notion of what can be substituted for and what can be substituted when applying substitution. Relation $\prec_i$ stores the time precedence between the frame and the input names, thus avoiding substituting a later emitted bounded name or term for a currently substitutable name.

Definition 6. *We say a pair of substitution $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2 \rangle$* respects *$e$, denoted by $\boldsymbol{\sigma} \rhd e$ if the following holds,*

- *$dom(\sigma_i) \subseteq V$ for $i \in \{1, 2\}$;*
- *$\forall x \in V : x \in dom(\sigma_1) \iff x \in dom(\sigma_2)$;*
- *$\forall x \in V : x\sigma_1 = x\sigma_2$, where $\phi_i \prec_i x$ for $i \in \{1, 2\}$;*
- *if $x \in Chan$, then $x\sigma_i \in N$ for $i \in \{1, 2\}$.*

The environment should be updated when a pair of substitution is applied, reflecting the changes of names.

Definition 7. *Let $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2 \rangle$ be a pair of substitutions, $e = \langle V, \prec \rangle$ be an environment, the* updated *environment $e^{\boldsymbol{\sigma}} = \langle V', \prec' \rangle$ is defined as follows,*

- *$V' = V \setminus dom(\boldsymbol{\sigma})$;*
- *$\phi_i\sigma_i \prec'_i x\sigma_i$ if $\phi_i \prec_i x$.*

The environment should also be extended when an input bounded name becomes free, e.g. the x in P when $a(x).P \xrightarrow{a(x)} P$. So we have the following definition.

Definition 8. *Let $e = \langle V, \prec \rangle$ be an environment. The* extension *of e, denoted by $e \oplus_V (x) = \langle V', \prec \rangle$ will be defined as,*

- *$V' = V \cup \{x\}$;*
- *$\prec'_i = \prec_i \cup \{\langle \phi_i, x \rangle\}$.*

We define $e^{-1} = \langle V^{-1}, \prec^{-1} \rangle$ where $\prec^{-1}$ is the reverse relation of $\prec$. An *open* relation R is a subset of $E \times P \times P$, such that $\forall \langle e, P_1, P_2 \rangle \in R : fn(P_i) \subseteq V$. It is called *symmetric* if $\forall \langle e, P_1, P_2 \rangle \in R$: $\langle e^{-1}, P_2, P_1 \rangle \in R$.

Definition 9. *An open relation R is an* open bisimulation *if $\langle e, P_1, P_2\rangle \in R$ implies $P_1 \approx_s P_2$, and for all $\boldsymbol{\sigma} \rhd e$, if $P_1\sigma_1 \xrightarrow{\mu_1} P_1'$ with $bn(\mu_1) \cap fn(P_1) = \emptyset$, then there exist P_2', μ_2 such that $P_2\sigma_2 \xrightarrow{\tau}^* \xrightarrow{\mu_2} \xrightarrow{\tau}^* P_2'$ with $bn(\mu_2) \cap fn(P_2) = \emptyset$ and,*

- *if $\mu_1 = \tau$, then $\mu_2 = \tau$ and $\langle e^{\boldsymbol{\sigma}}, P_1', P_2'\rangle \in R$;*
- *if $\mu_1 = \bar{a}\langle x\rangle$, then $\mu_2 = \bar{a}\langle x\rangle$ and $\langle e^{\boldsymbol{\sigma}}, P_1', P_2'\rangle \in R$;*
- *if $\mu_1 = a(M_1)$, then $\mu_2 = a(M_2)$ where $M_1 = M_2$ and $\langle e^{\boldsymbol{\sigma}} \oplus_V (fn(M_1, M_2)), P_1', P_2'\rangle \in R$.*

We say that P and Q are open bisimilar, denoted by $P \approx_o^e Q$, if there exists an open bisimulation R such that $\langle e, P, Q\rangle \in R$. We call $\approx_o^{\langle fn(P,Q),\emptyset,\emptyset\rangle}$ open bisimilarity.

Our definition of open bisimilarity is sound with respect to labeled bisimilarity.

Theorem 1. *If $P \approx_o^{\langle fn(P,Q),\langle\emptyset,\emptyset\rangle\rangle} Q$ then $P \approx_l Q$.*

The example stated in Section 1 shows that there exist two labeled bisimilar processes which are not open bisimilar. So we have the following corollary.

Corollary 1. *Open bisimilarity is strictly included in labeled bisimilarity.*

Congruence is an important property of equivalence relation in process algebra. It requires an equivalent relation closed under every operator. A congruent equivalence relation means we can verify security properties of a large system by break up to small parts, then each small part satisfy a security property implies the whole system also satisfy it. Fortunately, we have the following theorem.

Theorem 2. *Open bisimilarity is a congruence.*

3 Analyzing iKP Anonymity

iKP (i-Key-Protocol, i = 1, 2, 3) is a family of secure electronic payment protocols [19,20]. It involves three parties: the buyer B, the seller S and the acquirer A. We are concerned in the anonymity of 1KP.

We write SK_X for the secret key of party X ($X \in \{B, S, A\}$), and $PK(SK_X)$ for its corresponding public key. H is an ideal hash function. E_X means encryption and S_X means signature. We will use the following notations for messages transmitted in the protocol:

- $Desc$: Description of purchase and delivery address.
- $Salt_B$: Random number to salt $Desc$.
- $Authprice$: Amount and currency.
- $Date$: Time stamp.
- $Nonce_S$: Nonce of S.
- ID_S: ID of S.
- TID_S: Transaction ID chosen by the seller.
- Ban_B: Buyer's account number.
- R_B: Random number chosen by B.

- ID_B: Pseudo-ID of B, which is equal to $H(R_B, Ban_B)$.
- $Code$: Authorization code.

Composite messages are defined as follows:

- $Common = (Authprice, ID_S, TID_S, Date, Nonce_S, ID_B, H(Salt_B, Desc))$.
- $Clear = (ID_S, TID_S, Date, Nonce_S, H(Common))$.
- $Slip = (Authprice, H(Common), Ban_B, R_B)$.
- $EncSlip = E_A(Slip)$.
- $Sig_A = S_A(Code, H(Common))$.

The operation flow of this protocol consists of six steps:

1. **Initiate** $B \longrightarrow S : Salt_B, ID_B$. The buyer sends init, which is composed of the hash code of his/her account number, a random number, etc.
2. **Invoice** $S \longrightarrow B : Clear$. The seller answers by the hash code of price and other information. The message is composed of common and clear.
3. **Payment** $B \longrightarrow S : EncSlip$. The buyer checks the invoice and sends the payment slip encrypted by the acquirer's public key.
4. **Auth-Request** $S \longrightarrow A : Clear, H(Salt_B, Desc), EncSlip$. The seller requests the acquirer to authorize the payment. The message is composed of clear and req.
5. **Auth-Response** $A \longrightarrow S : Code, Sig_A$. The acquirer checks the request, extracts information from it, authorizes it and then sends response resp to the seller.
6. **Confirm** $S \longrightarrow B : Code, Sig_A$. The seller extracts the response and the signature of the acquirer, verifies it and then sends confirm to the buyer.

We now model this protocol in Applied Pi. The signature Σ of it is:

$$\Sigma = \{\mathsf{init}, \mathsf{clear}, \mathsf{common}, \mathsf{slip}, \mathsf{req}, \mathsf{resp}, \mathsf{confirm}, \mathsf{proj.i}, \\ \mathsf{hash}, \mathsf{dec}, \mathsf{enc}, \mathsf{checksig}, \mathsf{pk}, \mathsf{true}, \mathsf{false}\}$$

The function symbols init, clear, common, slip, req, resp, and confirm represent the message flows transmitted in the protocol. Their parts can be extracted. The extraction operator here we use is proj.i. The rest of function symbols in Σ is self-evident. They are common cryptographic primitives.

The equational theory is:

$$\begin{aligned} \mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(z)), z) &= x \\ \mathsf{checksig}(\mathsf{enc}(x, z), \mathsf{pk}(z)) &= \mathsf{true} \\ \mathsf{proj.i}(tuple(x_1, \ldots, x_l)) &= x_i \qquad \text{if } 1 \le i \le l \end{aligned}$$

The last equation is a short form of a family of equations with similar forms, that is

$$tuple \in \{\mathsf{init}, \mathsf{clear}, \mathsf{common}, \mathsf{slip}, \mathsf{req}, \mathsf{resp}, \mathsf{confirm}\}$$

We use substitutions to define the messages transmitted in the protocol:

$$\begin{aligned}
\sigma_1 &= \{\mathsf{init}(Salt_B, ID_B)/x_1\}\\
\sigma_2 &= \{\mathsf{clear}(ID_S, TID_S, Date, Nonce_S,\\
&\qquad \mathsf{hash}(Authprice, ID_S, TID_S, Date, Nonce_S,\\
&\qquad\quad \mathsf{proj.2}(x_1), \mathsf{hash}(\mathsf{proj.1}(x_1), Desc)))/x_2\}\\
\sigma_3 &= \{\mathsf{enc}(\mathsf{slip}, \mathsf{pk}(SK_A))/x_3\}\\
\sigma_4 &= \{\mathsf{req}(x_2, \mathsf{hash}(\mathsf{proj.1}(x_1), Desc), x_3)/x_4\}\\
\sigma_5 &= \{\mathsf{resp}(Code, \mathsf{enc}(Code, \mathsf{hash}(\mathsf{common}), SK_A))/x_5\}\\
\sigma_6 &= \{\mathsf{confirm}(\mathsf{proj.1}(x_5), \mathsf{proj.2}(x_5))/x_6\}
\end{aligned}$$

This approach is inspired by Gu *et al* [16].

Three parties are defined as follows: P_B stands for the buyer, P_S for the seller and P_A for the acquirer. In addition, $F_B(x), F_S(x), F_A(x)$ represent respectively the actions which the buyer, the seller and the acquirer are required to perform after finishing the protocol flows. The processes are:

$$\begin{aligned}
P &\equiv (\nu Authprice, Desc).P_B \mid P_S \mid P_A\\
P_B &\equiv (\nu Salt_B, R_B).(\overline{C_{BS}}\langle x_1\sigma_1\rangle \mid !(C_{SB}(x_2).\\
&\quad \mathsf{if}\ \mathsf{proj.5}(x_2) = \mathsf{hash}(Authprice, \mathsf{proj.1}(x_2), \mathsf{proj.2}(x_2),\\
&\qquad \mathsf{proj.3}(x_2), \mathsf{hash}(R_B, Ban_B), \mathsf{hash}(Salt_B, Desc))\\
&\quad \mathsf{then}\ (\overline{C_{BS}}\langle x_3\sigma_3\rangle \mid !(C_{SB}(x_6).\mathsf{if}\ \mathsf{checksig}(\mathsf{proj.2}(x_6),\\
&\qquad \mathsf{pk}(SK_A))\ \mathsf{then}\ F_B(x_6)))))\\
P_S &\equiv (\nu ID_S, TID_S, Date, Nonce_S).(C_{BS}(x_1).(\overline{C_{SB}}\langle x_2\sigma_2\rangle\\
&\quad \mid C_{BS}(x_3).(\overline{C_{SA}}\langle x_4\sigma_4\rangle \mid C_{AS}(x_5).\overline{C_{SB}}\langle x_6\sigma_6\rangle.F_S(x_5))))\\
P_A &\equiv (\nu SK_A).!(C_{SA}(x_4).\mathsf{if}\ \mathsf{proj.2}(\mathsf{dec}(\mathsf{proj.3}(x_4), SK_A)) =\\
&\quad \mathsf{hash}(\mathsf{proj.1}(\mathsf{dec}(\mathsf{proj.3}(x_4), SK_A)), \mathsf{proj.1}(\mathsf{proj.1}(x_4)),\\
&\qquad \mathsf{proj.2}(\mathsf{proj.1}(x_4)), \mathsf{proj.3}(\mathsf{proj.1}(x_4)), \mathsf{proj.4}(\mathsf{proj.1}(x_4)),\\
&\qquad\quad \mathsf{hash}(\mathsf{proj.3}(\mathsf{dec}(\mathsf{proj.3}(x_4), SK_A)), \mathsf{proj.4}(\mathsf{dec}(\mathsf{proj.3}(x_4),\\
&\qquad\qquad SK_A))), \mathsf{proj.1}(x_4))\ \mathsf{then}\ (\nu Code).(\overline{C_{AS}}\langle x_5\sigma_5\rangle.F_A(x_4)))
\end{aligned}$$

Buyers want to keep anonymous from eavesdroppers and sellers. Furthermore, buyers may even want to keep anonymous from the payment system provider (acquirers). We will show that iKP does not offer anonymity with respect to the payment system provider. It does minimize the exposure of buyers' identities to sellers and eavesdroppers. The crucial point is whether the Ban_B (Buyer B's Account Number) in P_B can be exposed to the environment. The results are formally stated in the following theorem.

Theorem 3. *Suppose $F_A(x_1) \approx_o^e F_A(x_2)$ for every e if and only if $x_1 = x_2$ and the same property holds for $F_B(x)$ and $F_S(x)$. Suppose $P_B(x)$ be the process P_B having x substituted for Ban_B and $B_1 \neq B_2$. Let $v = \{C_{SB}, C_{BS}\}$,*

$v' = v \cup \{C_{AS}, C_{SA}\}$ and $p = \langle \emptyset, \emptyset \rangle$. If we abbreviate $(\nu Authprice, Desc).$ $P_B(x) \,|\, P_S$ to $P_{BS}(x)$, then we have

$$P_{BS}(B_1) \approx_o^{\langle v,p \rangle} P_{BS}(B_2)$$

and

$$P_{BS}(B_1) \,|\, P_A \not\approx_o^{\langle v',p \rangle} P_{BS}(B_2) \,|\, P_A$$

Proof (sketch). Let us play the bisimulation game using our open approach. In each step, we update or extend the environment accordingly in order to record the knowledge exposed to the environment faithfully. We find that Ban_B only occurs in the frame of processes. After $P_{BS}(B_1)$ and $P_{BS}(B_2)$ have fired their respective output actions $\overline{C_{BS}}\langle x_3 \sigma_3 \rangle$,

$$\phi(P_{BS}(B_1)) = \{\mathsf{enc}((Authprice, \mathsf{hash}(\mathsf{common}), B_1, R_B), \mathsf{pk}(SK_A))/x_3\} \\ \,|\, \{(Salt_B, \mathsf{hash}(R_B, B_1))/x_1\} \,|\, \sigma_1 \,|\, \sigma_2 \,|\, \sigma_3$$

and

$$\phi(P_{BS}(B_2)) = \{\mathsf{enc}((Authprice, \mathsf{hash}(\mathsf{common}), B_2, R_B), \mathsf{pk}(SK_A))/x_3\} \\ \,|\, \{(Salt_B, \mathsf{hash}(R_B, B_2))/x_1\} \,|\, \sigma_1 \,|\, \sigma_2 \,|\, \sigma_3$$

It is obvious that $\phi(P_{BS}(B_1)) \approx_s \phi(P_{BS}(B_2))$, then $P_{BS}(B_1) \approx_o^{\langle v,p \rangle} P_{BS}(B_2)$ follows easily.

For the case in which acquirers are taken account of, $\mathsf{proj.3}(\mathsf{dec}(x_3, SK_A)) = B_1$ is deducible in $\phi(P_{SB}(B_1) \,|\, P_A)$, but is not deducible in $\phi(P_{SB}(B_2) \,|\, P_A)$. Therefore we have $\phi(P_{BS}(B_1) \,|\, P_A) \not\approx_s \phi(P_{BS}(B_2) \,|\, P_A)$. This suggests us that the attacker hidden in the environment can tell the difference of two account numbers. By the assumption that $F_A(x_1) \not\approx_o^e F_A(x_2)$ for all e if $x_1 \neq x_2$, we have $P_{SB}(B_1) \,|\, P_A \not\approx_o^{\langle v',p \rangle} P_{BS}(B_2) \,|\, P_A$. □

4 Conclusion

In this paper we propose a knowledge aware open bisimulation relation for the Applied Pi Calculus. Our relation is more suitable for analyzing security protocols. It also has many other advantages, which include the congruent property and a finer distinguishability. By analyzing the anonymity of iKP, it is shown that playing the knowledge aware bisimulation game can effectively analyze anonymity, which is hard to obtain by many other methods.

As for the future work, we would like to apply this approach to check other security properties, such as authentication and non-repudiation. We are also working on developing automatically equivalence checking tools.

Acknowledgements

The authors would like to thank anonymous referees for their constructive comments and suggestions.

References

1. Abadi, M., Fournet, C.: Mobile Values, New Names, and Secure Communication. In: Proceedings of POPL 2001, pp. 104–115. ACM Press, New York (2001)
2. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, part I. Information and Computation 100(1), 1–40 (1992)
3. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, part II. Information and Computation 100(1), 41–77 (1992)
4. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the Pi Calculus. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 340–354. Springer, Heidelberg (2004)
5. Abadi, M., Blanchet, B.: Computer-Assisted Verification of a Protocol for Certified Email. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 316–335. Springer, Heidelberg (2003)
6. Fournet, C., Abadi, M.: Hiding Names: Private Authentication in the Applied pi Calculus. In: Okada, M., Pierce, B.C., Scedrov, A., Tokuda, H., Yonezawa, A. (eds.) ISSS 2002. LNCS, vol. 2609, pp. 317–338. Springer, Heidelberg (2003)
7. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. Software - Concepts and Tools 17(3), 93–102 (1996)
8. Abadi, M., Gordon, A.: A Calculus for Cryptographic Protocols: The Spi Calculus. Information and Computation 148, 1–70 (1999)
9. Abadi, M., Gordon, A.: A Bisimulation Method for Cryptographic Protocols. Nordic Journal of Computing 5(4), 267–303 (1998)
10. Borgström, J., Briais, S., Nestmann, U.: Symbolic Bisimulation in the Spi Calculus. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 161–176. Springer, Heidelberg (2004)
11. Chaum, D.: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. Journal of Cryptology 1, 65–75 (1988)
12. Schneider, S., Sidiropoulos, A.: CSP and Anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146. Springer, Heidelberg (1996)
13. Kremer, S., Ryan, M.: Analysis of an Electronic Voting Protocol in the Applied Pi-Calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)
14. Chothia, T.: Analysing the MUTE Anonymous File-Sharing System using the pi-Calculus. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229. Springer, Heidelberg (2006)
15. Cai, X., Gu, Y.: Measuring anonymity. In: 5th Information Security Practice and Experience Conference (ISPEC 2009). LNCS, vol. 5451, pp. 183–194. Springer, Heidelberg (2009)
16. Gu, Y., Li, G., Fu, Y.: Analyzing iKP Security in Applied pi Calculus. In: Zhang, J., He, J.-H., Fu, Y. (eds.) CIS 2004. LNCS, vol. 3314. Springer, Heidelberg (2004)
17. Zhu, H., Cai, X.: An Open Approach for the Applied pi Calculus. Technical report, Shanghai Jiao Tong University (2008)
18. Abadi, M., Cortier, V.: Deciding Knowledge in Security Protocols under Equational Theories. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 46–58. Springer, Heidelberg (2004)
19. Bellare, M., Garay, J., Hauser, R., et al.: Design, Implementation, and Deployment of the iKP Secure Electronic Payment System. IEEE Journal of Selected Areas in Communications 18, 611–627 (2000)
20. Bellare, M., Garay, J., et al.: iKP — A Family of Secure Electronic Payment Protocols. In: Proceedings of USENIX Workshop on Electronic Commerce (1995)

A Parallel High-Order Discontinuous Galerkin Shallow Water Model

Claes Eskilsson[1], Yaakoub El-Khamra[1], David Rideout[2], Gabrielle Allen[1], Q. Jim Chen[1], and Mayank Tyagi[1]

[1] Louisiana State University, Baton Rouge LA 70803, USA
[2] Perimeter Institute for Theoretical Physics, Waterloo Ontario N2L 2Y5, Canada

Abstract. The depth-integrated shallow water equations are frequently used for simulating geophysical flows, such as storm-surges, tsunamis and river flooding. In this paper a parallel shallow water solver using an unstructured high-order discontinuous Galerkin method is presented. The spatial discretization of the model is based on the Nektar++ spectral/*hp* library and the model is numerically shown to exhibit the expected exponential convergence. The parallelism of the model has been achieved within the Cactus Framework. The model has so far been executed successfully on up to 128 cores and it is shown that both weak and strong scaling are largely independent of the spatial order of the scheme. Results are also presented for the wave flume interaction with five upright cylinders.

1 Introduction

This paper presents a first step towards a community coastal modeling toolkit for nearshore surface water waves based on spectral/*hp* element methods. The ultimate goal is a scalable, parallel, non-hydrostatic wave solver, based on multi-layered Boussinesq-type equations including time-dependent bathymetry and sediment transport. In this paper we outline the ongoing work of the parallel implementation of non-dispersive two-dimensional shallow water equations (SWE). As Boussinesq-type equations are higher-order extensions to the SWE, the SWE constitute the natural initial stepping stone. A driving force behind this effort is storm surge modeling and the SWE model presented here can be used as an efficient hydrodynamic core for such simulations.

There are several motivations for using spectral/*hp* element methods (i.e. finite element methods of arbitrarily high order) rather than more traditional methods. Spectral/*hp* elements provide a flexible setting where both the element size and the polynomial order within the elements can be altered. Further, high-order methods tend to be more computationally efficient for long-time integrations compared to low-order methods due to their inherently small numerical diffusion/dispersion error. The discontinuous Galerkin (DG) flavour of spectral/*hp* elements was choosen mainly since the resulting global mass matrix is block-diagonal. Thus, any explicit time-stepping scheme can update the numerical solution in an element-by-element fashion. The DG method is also able to incorporate well established shock-capturing techniques from the finite volume (FV) framework.

SWE models are most frequently based on shock-capturing FV methods, but over the last decade several DG SWE models have been presented. Later studies emphasize the use of high-order schmes [5,4,11,6] and the present state-of-the-art DG SWE models rely on adaptivity: [1] include h-type adaptivity while [13] illustrates the benefits of p-type adaptivity. In contrast to a recent study [12] that addresses the parallel performance of a low-order DG SWE model, the present study is addressing the performance of high-order schemes.

The coastal DG wave model is based on the spectral/*hp* element library Nektar++ [9,10], while parallelism has been achieved by adding support for unstructured two-dimensional meshes into the computational framework Cactus [8,7]. This separation of programming tasks allows coastal engineers to focus on developing coastal code using Nektar++ and the computational scientists to focus on parallelism, performance and scalability of the unstructured mesh driver. Further, the adoption of an underlying parallel framework provides a methodology to develop interoperable modules to add additional solvers and models leading to a comprehensive toolkit for modeling coastal environments.

2 Shallow Water Equations

The shallow water equations are a set of non-linear hyperbolic equations. The equations are derived under the assumption of hydrostatic pressure, and thus the SWE are only valid for long waves (the rule of thumb being that the still water depth to wavelength ratio should be less than 1/20). The conservation form of the SWE read

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = \mathbf{S}(\mathbf{U}) \,, \tag{1}$$

where $\mathbf{U} = [H \,, Hu \,, Hv]^{\mathrm{T}}$ is the vector of conserved variables and $\mathbf{F}(\mathbf{U}) = [\mathbf{E}(\mathbf{U}) \,, \mathbf{G}(\mathbf{U})]^{\mathrm{T}}$ is the flux vector defined by:

$$\mathbf{E} = \begin{bmatrix} Hu \\ Hu^2 + gH^2/2 \\ Huv \end{bmatrix} , \qquad \mathbf{G} = \begin{bmatrix} Hv \\ Huv \\ Hv^2 + gH^2/2 \end{bmatrix} .$$

Here $H(\mathbf{x},t) = \eta(\mathbf{x},t) + d(\mathbf{x})$ is the total water depth, $\eta(\mathbf{x},t)$ is the free surface elevation and $d(\mathbf{x})$ is the still water depth; $\mathbf{u}(\mathbf{x},t) = [u(\mathbf{x},t) \,, v(\mathbf{x},t)]^{\mathrm{T}}$ denotes the depth-averaged velocity in the x- and y-direction, respectively, while g is the acceleration of gravity.

The source terms $\mathbf{S}(\mathbf{U})$ can contain forcing due to e.g. bathymetry, bottom friction, atmospheric pressure, Coriolis force, wind stresses and diffusion. The objective of this study is to assess the performance of the computational core rather than present any solution of real-life engineering cases, and we focus here on the homogeneous version of the SWE.

3 Numerical Scheme

3.1 DG Discretization

Let $\varOmega_h$ denote the partition of the computational domain, $\varOmega$, into N non-overlapping elements $\varOmega_e$ with elemental boundaries $\varGamma_e$. The diameter of the element $\varOmega_e$ is given by h_e and subsequently $h = \max(h_1, \ldots, h_N)$. We introduce the discrete polynomial space

$$\mathcal{V}_\delta = \left\{ v \in L^2(\varOmega) \,:\, v|_{\varOmega_e} \in \mathcal{P}^p(\varOmega_e), \forall \varOmega_e \in \varOmega_h \right\} ,$$

where $\mathcal{P}^p$ is the space of polynomials of degree at most p in the element $\varOmega_e$.

We proceed by multiplying eq. (1) with a piecewise smooth test function $q(\mathbf{x})$ and integrate over the local element $\varOmega_e$. We then approximate $\mathbf{U}$ and q with polynomial expansions of order p:

$$\int_{\varOmega_e} q_\delta \frac{\partial \mathbf{U}_\delta}{\partial t}\, d\mathbf{x} + \int_{\varOmega_e} q_\delta \nabla \cdot \mathbf{F}(\mathbf{U}_\delta)\, d\mathbf{x} = \int_{\varOmega_e} q_\delta\, \mathbf{S}(\mathbf{U}_\delta)\, d\mathbf{x} .$$

After applying the divergence theorem and exchanging the boundary flux term with a numerical flux, we can state the discrete DG method as: find $\mathbf{U}_\delta \in \mathcal{V}_\delta$ such that for all $q_\delta \in \mathcal{V}_\delta$ and for all $\varOmega_e \in \varOmega_h$

$$\begin{aligned} \int_{\varOmega_e} q_\delta \frac{\partial \mathbf{U}_\delta}{\partial t}\, d\mathbf{x} - \int_{\varOmega_e} \nabla q_\delta \cdot \mathbf{F}(\mathbf{U}_\delta)\, d\mathbf{x} \\ + \int_{\varGamma_e} q_\delta\, \hat{\mathbf{F}}(\mathbf{U}_\delta) \cdot \mathbf{n}\, dS = \int_{\varOmega_e} q_\delta\, \mathbf{S}(\mathbf{U}_\delta)\, d\mathbf{x} , \end{aligned} \tag{2}$$

where $\mathbf{n}$ is the outward unit normal to $\varGamma_e$ and $\hat{\mathbf{F}}$ denotes the continuous numerical flux used to couple the elements together. In this study we use the popular HLLC Riemann solver [14] as the numerical flux.

The approximation of an arbitrary variable $f_\delta \in \mathcal{V}_\delta$ in the local element $\varOmega_e$ can be expressed as

$$f_\delta(\mathbf{x}, t) = \sum_{i=0}^{N^e_{\text{dof}}-1} \tilde{\mathbf{f}}_e[i] \phi_i(\mathbf{x}) , \qquad \mathbf{x} \in \varOmega_e ,$$

where $\tilde{\mathbf{f}}_e[i]$ is the time-dependent vector consisting of the N^e_{dof} elemental degrees of freedom of expansion coefficients and $\phi_i(\mathbf{x})$ are the trial functions.

Introducing the elemental mass matrix

$$\mathbf{M}^e[p][q] = \int_{\varOmega_e} \phi^e_p(\mathbf{x})\, \phi^e_q(\mathbf{x})\, d\mathbf{x} ,$$

and the elemental weak derivative matrices

$$\mathbf{D}^e_x[p][q] = \int_{\varOmega_e} \frac{\partial \phi^e_p(\mathbf{x})}{\partial x} \phi^e_q(\mathbf{x})\, d\mathbf{x} , \qquad \mathbf{D}^e_y[p][q] = \int_{\varOmega_e} \frac{\partial \phi^e_p(\mathbf{x})}{\partial y}\, \phi^e_q(\mathbf{x})\, d\mathbf{x} ,$$

where $0 \leq p, q \leq N^e_{\text{dof}} - 1$, we can write eq. (2) in elemental form as

$$\mathbf{M}^e \frac{\partial \tilde{\mathbf{U}}}{\partial t} - \mathbf{D}^e_x \, \mathbf{E}(\tilde{\mathbf{U}}) - \mathbf{D}^e_y \, \mathbf{G}(\tilde{\mathbf{U}}) + \mathbf{f}^e = \mathbf{M}^e \, \mathbf{S}(\tilde{\mathbf{U}}) \, ,$$

in which

$$\mathbf{f}^e[p] = \int_{\Gamma_e} \phi_p(\mathbf{x}) \left(\hat{\mathbf{F}}(\mathbf{U}_\delta) \cdot \mathbf{n} \right) dS \, .$$

In contrast to continuous Galerkin methods, for DG methods the global matrices are just a concatenation of local elemental matrices. Hence, using explicit time-stepping schemes the solution can be computed in an element-by-element fashion.

Following the standard Galerkin formulation we use the same functions for the test and trial functions. Former studies concerned with high-order DG methods for SWE have primarily used either the modal orthogonal Prioli-Koornwinder-Dubiner (PKD) basis [4,11] or the nodal electrostatic basis [5,6]. Here we use the C^0 modified PKD basis [10], which can be decomposed into boundary and interior modes. For this choice of basis the elemental mass matrix is sparse, but on the other hand there are only the boundary modes involved in the computation of the boundary integral. More importantly this choice of expansion basis gives the possibility to use static condensation if a global equation system must be solved [10]. Admittedly, the present SWE solver does not involve a global solve, but this should be an important feature in the future development of a Boussinesq solver.

The semi-discretized equations are advanced in time using a second- or third-order TVD Runge-Kutta method and all boundary conditions are enforced weakly through the use of the Riemann solver.

3.2 Computational Approach

The discontinuous Galerkin scheme outlined in section 3.1 was first implemented in a stand-alone serial code using the open-source spectral/*hp* library Nektar++ [9]. Nektar++ provides the fundamental tools associated with a high-order finite element method, such as the calculation of expansion functions, inner products and differentiation. With regard to the high-order discretization our work has been focused on implementing a solver structure for time-dependent problems. This includes a SWE class containing functions for the evaluation of the flux vector, numerical fluxes, equation dependent boundary condition, various source terms, etc.

We rely on the Cactus Framework [8] to provide parallelization and to adopt an extensible component approach to code development. Cactus is an open source problem solving environment designed for scientists and engineers needing to develop collaborative code for large scale parallel machines. Cactus comprises sets of components (or *thorns*) which are invoked by the Cactus *Flesh* which collects information from the thorns to define grid variables, parameters, methods and scheduling. The modular structure of Cactus enables parallel computation

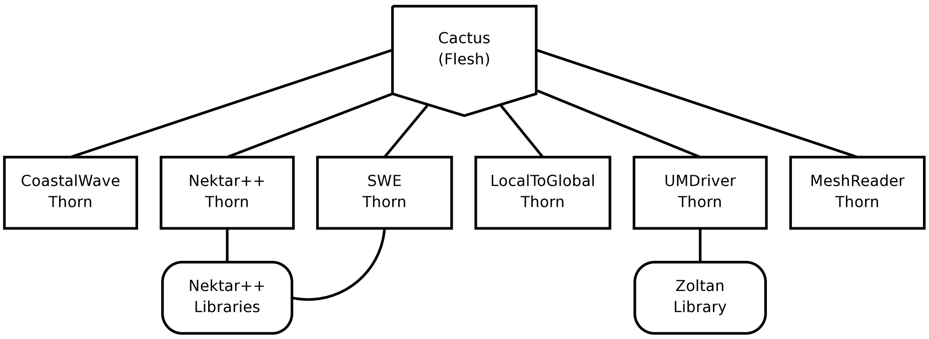


Fig. 1. Architectural diagram showing Cactus thorns and their connection to the external Nektar++ and Zoltan packages

across different architectures and collaborative code development between different groups.

To integrate the serial SWE solver into Cactus several new thorns were developed (see Figure 1). First, an unstructured mesh driver (thorn `UMDriver`) was developed which provides the underlying parallel layer and the Cactus Configuration Language was extended to support grid functions on unstructured meshes. The UMDriver (which is still under development) uses the Zoltan [3] library to provide mesh partitioning, load balancing and mesh migration. Another thorn `LocalToGlobal` provides local reindexing of elements, edges and vertices.

The core thorn for the coastal modeling toolkit in Cactus is `CoastalWave`. Mirroring the methodology for other domain specific toolkits in Cactus, this thorn defines the generic variables, parameters, and methods for coastal models, allowing models providing the same functionality to be swapped (e.g. the SWE will be interoperable with the Boussinesq solver) and allowing additional components to be added into the workflow (e.g. phase-averaging wave models to compute radiation stresses).

Thorn `Nektar++` initializes and populates the data structures of the Nektar++ library. Thorn `SWE` thorn contains the actual SWE solver based on routines defined in the SWE class. Finally, thorn `MeshReader` provides a simple ASCII mesh file reader, and also allows users to register their own mesh readers.

4 Computational Results

4.1 Convergence

Consider the simple case of a linear standing wave with a wavelength of 10 m in a square 10 by 10 m basin. The still water depth is 0.5 m. In order to compare with the analytical solution we here use the linearized SWE. We compute the solution for one wave period using 10 000 time steps. Table 1 shows the error and order of convergence measured in the L_2 norm.

Table 1. L_2 error and order of convergence for the linear standing wave

	$N = 16$	$N = 64$		$N = 256$	
p	error	error	order	error	order
1	3.3676E-03	6.4677E-04	2.38	1.6054E-04	2.01
2	1.7754E-04	2.1496E-05	3.05	2.5909E-06	3.05
3	2.1303E-05	2.2351E-06	3.25	6.3190E-08	5.14
4	3.4041E-07	9.7791E-09	5.12	3.0563E-10	5.00

In general we have optimal convergence $\mathcal{O}(h^{p+1})$, although for $p = 3$ there is an instance of $\mathcal{O}(h^p)$ convergence. This is due to the use of a simplified numerical flux (componentwise averaging) in the linear scheme. The use of averaging is known to sometimes produce sub-optimal convergence for odd p. Nevertheless, for p-type refinement we obtain the expected exponential convergence, illustrated by the approximate straight lines in Figure 2.

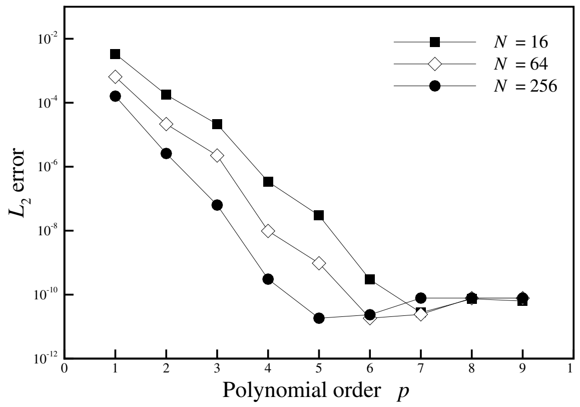


Fig. 2. Illustration of exponential convergence for the linear standing wave

4.2 Weak Scaling

Weak scaling indicates the ability of a code to scale up a problem on more cores, increasing the domain size or grid refinement while keeping a constant load on each core. We consider two series of meshes – consisting of 100 and 900 quadrilaterals per core, respectively. Both sets are run with three different polynomial orders: $p = 4, 6, 8$. The largest run (900 elements per core on 128 cores) thus contains 218 700 unique degrees of freedom per core, or a total of roughly 28 million degrees of freedom. We execute the model for one hundred time steps. Scaling tests were performed on the LONI "Queen Bee" Linux cluster (comprising 668 nodes, each containing dual Quad Core Xeon 64-bit 2.33GHz processors with an Infiniband interconnect).

Table 2. Weak scaling using 100 quadrilaterals [Top table] and 900 quadrilaterals [Bottom table] per core. Solution time is wall clock time in seconds while parallel efficiency is presented as an percentage.

Numbers	$p = 4$		$p = 6$		$p = 8$	
of cores	Solution time	Parallel eff.	Solution time	Parallel eff.	Solution time	Parallel eff.
1	5.00	—	7.01	—	10.6	—
2	5.56	89.8	7.77	90.3	11.7	90.6
4	6.16	81.2	8.54	82.1	12.7	83.4
8	7.17	69.7	9.98	70.3	14.7	72.1
16	7.53	66.3	10.35	67.8	15.3	69.0
32	7.75	64.5	10.63	66.0	16.0	66.1
64	8.44	59.2	11.45	61.3	16.5	64.0
128	9.06	55.2	12.13	57.8	17.2	61.4

Numbers	$p = 4$		$p = 6$		$p = 8$	
of cores	Solution time	Parallel eff.	Solution time	Parallel eff.	Solution time	Parallel eff.
1	134.3	—	153.1	—	185.7	—
2	140.6	95.6	160.1	95.6	193.7	95.8
4	150.5	89.3	169.5	90.3	203.5	91.3
8	159.8	84.1	181.9	84.1	218.6	85.0
16	167.4	80.3	185.9	82.3	222.5	83.5
32	165.0	81.4	190.0	80.6	226.0	82.2
64	167.1	80.4	194.5	78.7	227.8	81.5
128	170.6	78.7	191.4	80.0	237.6	78.2

Table 2 shows the solution time (without time spent in I/O, initializing and partitioning of the mesh) for the 100 and 900 element series. The parallel efficiency decreases for both series, although more rapidly for the 100 element series. This is due to the fact that ghost elements are currently used rather than ghost edges, so that the workload per core is not actually constant: a two-core partition has 100 plus an additional 10 ghost elements per core; four partitions has 100 elements plus 20 ghost elements per core, etc. This increase in computational overhead is especially pronounced for scaling tests with a low workload per core as well as for tests using less than eight cores. The parallel efficiency can be regarded as largely independent with regard to polynomial order p – the slight increase in efficiency for higher p can be attributed to the higher workload per core.

Scaling for the total execution time shows a larger decrease in efficiency, indicating that the initialization and mesh partitioning routines also require further attention to improve scalability. A core contributor to this effect is due to the fact that currently each core stores information about the indices on the entire mesh.

4.3 Strong Scaling

Strong scaling indicates the ability to decrease the total run time for a particular problem, scaling across more cores while keeping the overall problem size fixed.

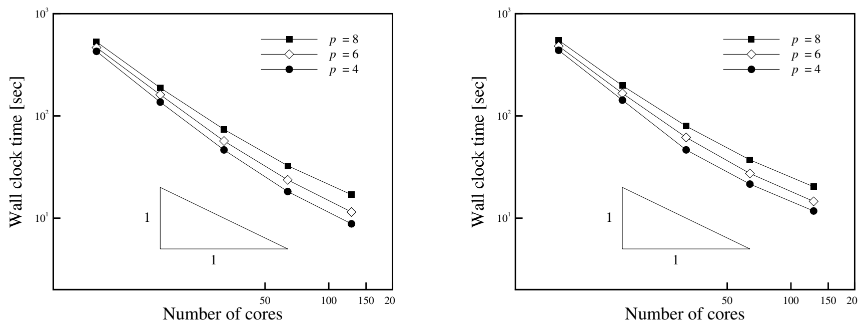


Fig. 3. Strong scaling. [Left] Solution time and [Right] total execution time.

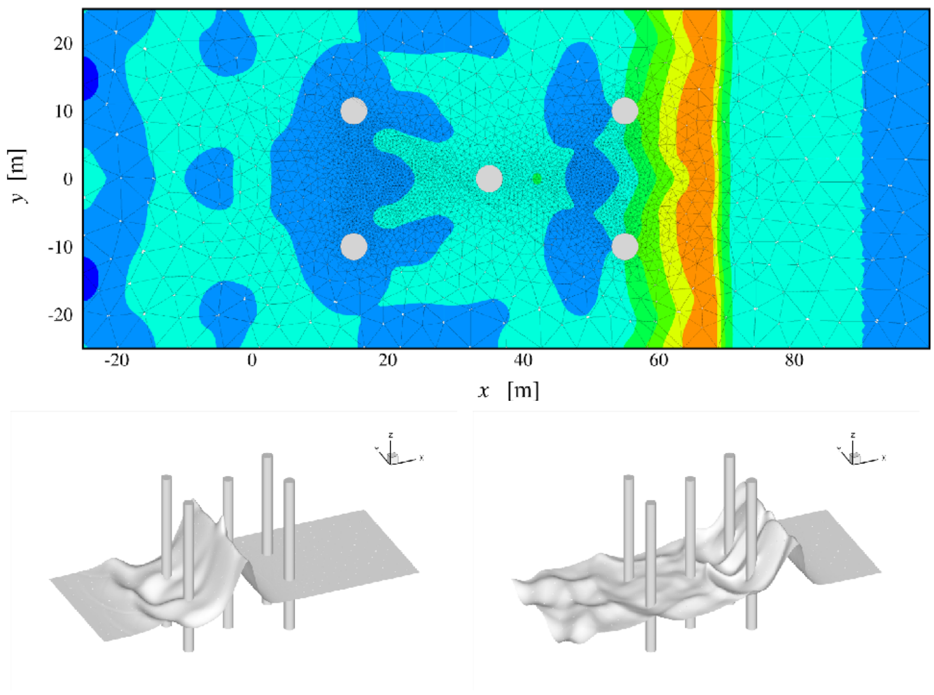


Fig. 4. Cylinder test case: [Top] computational mesh including iso-contours of the surface elevation at $t = 20$ seconds; [Bottom left] snapshot of surface elevation at $t = 9.9$ seconds and [Bottom right] snapshot of surface elevation at $t = 20$ seconds

We assess the strong scaling using a fixed size problem of 12 800 quadrilaterals. As for the weak scaling tests we use $p = 4, 6, 8$ and run the problem for 100 time steps. Figure 3 shows the reduction in solution time as the number of processors are increased. The curves in Figure 3 asymptotically approaches the 1:1 slope, indicating that the wall clock time is halved as the number of cores are doubled.

4.4 Wave Interacting with Cylinders

Consider a numerical wave flume that is 125 m long and 40 m wide, with an array of five cylinders (all having a 4 m diameter). The computational domain is discretized into roughly 6 000 triangular elements of polynomial order $p = 6$ (roughly 500 000 degrees of freedom), see the top image in Figure 4, heavily clustered around the cylinders. The initial condition is given by Laitone's first order solitary wave solution using an amplitude of 0.05 m centered at $x = 0$. We run the model for 25 seconds, using 10 000 steps. The non-linear SWE model is executed using 64 cores and the simulation generally takes 0.1 second per time step.

The lower images in Figure 4 shows the evolution of the solitary wave, including wave run-up, scattering, diffraction, reflection as well as interaction between the scattered waves. Note that the solution correctly remains symmetric throughout the simulation.

5 Concluding Remarks

We have outlined a spectral/*hp* DG method for solving the SWE and described it's integration with the Cactus Framework. This work concludes the first phase of a program to develop a scalable, parallel, non-hydrostatic wave solver, based on multi-layered Boussinesq-type equations capable of including time-dependent bathymetry and sediment transport. An important step has been taken in designing the integration of the underlying spectral/*hp* solver apparatus into Cactus to form the basis of a community framework for coastal modeling. This necessitated the development of an unstructured mesh driver for Cactus, which through the modular framework can now be used by other application domains.

Much work remains to be done. Comparing to the weak scaling results of e.g. [2] it is clear that the scaling of the SWE is not yet optimal. The main reasons for the suboptimal behaviour are: (i) additional communication overhead due to the use of complete ghost elements rather than just using ghost edges; (ii) current inefficiencies in the Cactus UMDriver thorn, e.g. in duplicating the physical space variables.

That the scaling was found independent of polynomial approximation is encouraging, as we can expect scalability also for higher-order schemes. Planned future work on the unstructured mesh driver includes improving scaling on large numbers of cores, adaptive mesh refinement, a hyperslabbing interface and dynamic load balancing. On the coastal modeling side, subroutines for e.g. flooding/drying and breaking waves will be added to allow for real-life engineering applications.

Acknowledgements

The authors gratefully acknowledge the contributions of Prof. S.J. Sherwin, Mr. P. Vos (Imperial College London) and Prof. R.M. Kirby (University of Utah) to the DG SWE solver. Financial support was provided by DOD/ONR

(N00014-07-1-0955), NSF/EPSCOR (EPS-0701491) and the Floating Point Systems Endowed Chair at LSU. This research was supported in part by the Perimeter Institute for Theoretical Physics. Computer resources were provided by the Louisiana Optical Network Initiative (LONI).

References

1. Bernard, P.-E., Chevaugeon, N., Legat, V., Deleersnijder, E., Remacle, J.-F.: High-order h-adaptive discontinuous Galerkin methods for ocean modelling. Ocean Dyn. 57, 109–121 (2007)
2. Biswas, R., Devine, K.D., Flaherty, J.: Parallel, adaptive finite element methods for conservation laws. Appl. Numer. Math. 14, 255–283 (1994)
3. Devine, K., Boman, E., Heaphy, R., Hendrickson, B., Vaughan, C.: Zoltan Data Management Services for Parallel Dynamic Applications. Comp. Sci. Eng. (2002)
4. Eskilsson, C., Sherwin, S.J.: A triangular spectral/*hp* discontinuous Galerkin method for modelling 2D shallow water equations. Int. J. Num. Meth. Fluids 45, 605–623 (2004)
5. Giraldo, F.X., Hesthaven, J.S., Warburton, T.: Nodal high-order discontinuous Galerkin methods for the spherical shallow water equations. J. Comp. Phys. 181, 499–525 (2002)
6. Giraldo, F.X., Warburton, T.: A high-order triangular discontinuous Galerkin oceanic shallow water model. Int. J. Num. Meth. Fluids 56, 899–925 (2008)
7. Goodale, T., Allen, G., Lanfermann, G., Massó, J., Radke, T., Seidel, E., Shalf, J.: The Cactus framework and toolkit: Design and applications. Vector and Parallel Processing. Springer, Heidelberg (2003)
8. `http://www.cactuscode.org`
9. `http://www.nektar.info`
10. Karniadakis, G.E., Sherwin, S.J.: Spectral/hp Element Methods for Computational Fluid Dynamics. Oxford Sci. Publ. (2005)
11. Kubatko, E.J., Westerink, J.J., Dawson, C.: *hp* discontinuous Galerkin methods for advection dominated problems in shallow water flow. Comp. Meth. Appl. Mech. Eng. 196, 437–451 (2006)
12. Kubatko, E.J., Bunya, S., Dawson, C., Westerink, J.J., Mirabito, C.: A performance comparison of continuous and discontinuous finite element shallow water models. J. Sci. Comp. (in press)
13. Kubatko, E.J., Bunya, S., Dawson, C., Westerink, J.J.: Dynamic p-adaptive Runge-Kutta discontinuous Galerkin methods for the shallow water equations. Comput. Meth. Appl. Mech. Engrg. (in press)
14. Toro, E.T.: Shock-Capturing Methods for Free-Surface Shallow Flows. John Wiley, Chichester (2001)

CelOWS: A Service Oriented Architecture to Define, Query and Reuse Biological Models

Ely Edison Matos[1], Fernanda Campos[1,2], Regina Braga[1,2], Rodrigo Weber[1,3], and Daniele Palazzi[1,2]

[1] Master Program in Computational Modeling
[2] Software Quality Research Group
[3] Computational Physiology Laboratory
Federal University of Juiz de Fora - Juiz de Fora – Minas Gerais – Brazil
{ely.matos,fernanda.campos,regina.braga, rodrigo.weber}@ufjf.edu.br, daniele.palazzi@ice.ufjf.br

Abstract. The amount of information generated by biological research has lead to an intensive use of models. Mathematical and computational modeling needs accurate description to share, reuse and simulate models as formulated by original authors. In this paper, we introduce the Cell Component Ontology - CelO, expressed in OWL-DL. This ontology captures both the structure of a cell model and the properties of functional components. We use this ontology in a Web project – CelOWS - to describe, query and compose CellML models. It aims to improve reuse and composition of existent components and allow semantic validation of new models.

Keywords: Semantic web services, ontology, SOA, e-Science.

1 Introduction

Intense research in biological science has generated great volume of data. Tools, methods and techniques improve the understanding of new functions, structures and processes related to biophysics and physiology. The increase of the computational power and the use of numerical methods stimulate the development and the application of more complex models [1]. These models allow the combination of different physical experiments and in different scales into a computational simulation, providing an accurate view of the studied phenomena.

The computational simulation of a model involves two important aspects. The first one is the model representation. Although diagrams, literal description and equations can be used to publish the models, typographical errors, as well as the necessary conditions to the simulation may generate fatal errors. The second aspect is concerned to the implementation phase. The necessity to apply advanced numerical methods creates a barrier for the effective use and study of the model.

These questions have stimulated the development of CellML [2], a markup language for representing biological models. Based on XML (eXtensible Markup Language), CellML specifies elements that can be used to represent a model in a formal way, without ambiguities, legible for humans and computer-readable. The mathematical equations are represented in MathML, which makes it independent of a specific implementation language. CellML can be used to represent, store and share models. However, the CellML approach does not provide mechanisms to make component reuse an easy going step.

This research is aimed at using and discussing ontologies and semantic rules for creation, validation, storage and sharing of biological models, through the service oriented architecture - CelOWS. It allows the storage, research, reuse, composition and execution of described models using the CelO (Cell Component Ontology) ontology. The main goal of the CelOWS is the composition of a new model from the reuse of different models from a specific repository. The CelO ontology, described in OWL-DL [3], aims to add semantics to the models. It allows the representation of the intrinsic knowledge, to improve its validation, reuse components from other models, automate some composition processes and develop model repositories where queries are semantically carried out.

The remainder of the paper is organized as follows. The next section discusses the background of the work. Section 3 presents the proposed CelO ontology and its main elements. Section 4 describes the CelOWS architecture, use cases, validation process and examples of application. Related works are presented in Section 5, while Section 6 concludes the paper.

2 Related Concepts

In this research context, biological models are related to mathematical and computational representation of some biological properties. In the field of physiology and electrophysiology the functional elements of the system are generally represented by an abstraction named “component”. Each component is a mathematical model that aims to represent the behavior of the biological element. The interaction between the biological elements is represented by the association of different components.

The electrophysiology models currently span from simple models of the electric activation based on polynomials to three-dimensional complex models [4]. The cell components models can be represented through diagrams and have a series of associated equations. Diagrams and text descriptions, however, can have typographical errors or incorrect initial or contour conditions to the simulation. These concerns were recently addressed via the development of the CellML language for representing biological models.

CellML establishes a standard format for defining and sharing biological models. The models are then represented as an interconnected network of components. Each component represents a biological element of interest as, for example, the cell membrane or an ionic channel. From the viewpoint of computational representation, a component is a unit that can interact, add or encapsulate other components.

A model receives a name and an unique identification (cmeta: id) used to identify the model URI. The mathematical equations are expressed in MathML and they describe the behavior of each component in the model. A variable is a nominated entity associated to a component, representing amounts used in the equations. An initial value and some attributes (units, public and private interfaces, etc.) can be associated to a variable. Connections between components are represented through the variables mapping.

3 Cell Component Ontology - CelO

Ontology can be described as a formal and explicit specification of a shared conceptualization [6] [7]. The ontology construction implies in acquiring the domain knowledge and collecting the appropriate information that formally defines the domain terms. The CelO ontology can be extended for the Biology area, but by now it focus on the sub-domain of cardiac electrophysiology. This restriction aims to diminish the complexity level and allow a gradual evaluation of the ontology, in terms of its completeness and functionality.

The specific goal of the CelO ontology is to describe the semantic of biological models. The use of the semantic to express the intrinsic model knowledge aims to improve its validation, reuse other models components, automate the processes of models composition and construct Web repositories with semantic queries. The structure of the ontology aims its integration with simulations of CellML models, using tools as AGOS [8] and PCEnv [9].

For the development of the CelO ontology the Methontolgy methodology was adopted, which includes four phases [10]: specification, conceptualization, formalization and implementation. In the specification activity a document was defined with the ontology goals and main terms. During the conceptualization phase a set of representations organized the domain knowledge: terms glossary, concepts classification tree, diagrams of binary relationship and concepts dictionary. The activities of formalization and implementation phases were developed and the ontology built using the Protegé tool. The maintenance and evolution will occur whenever modifications in the ontology structure are necessary (Fig. 1)

The CelO ontology has three general classes in its top level and its structure provides three essential types of knowledge: SIEntity (quantities and units associated to variables of the model), DomainEntity (Concepts of the Biological domain) and ModelEntity (Model components and interfaces).

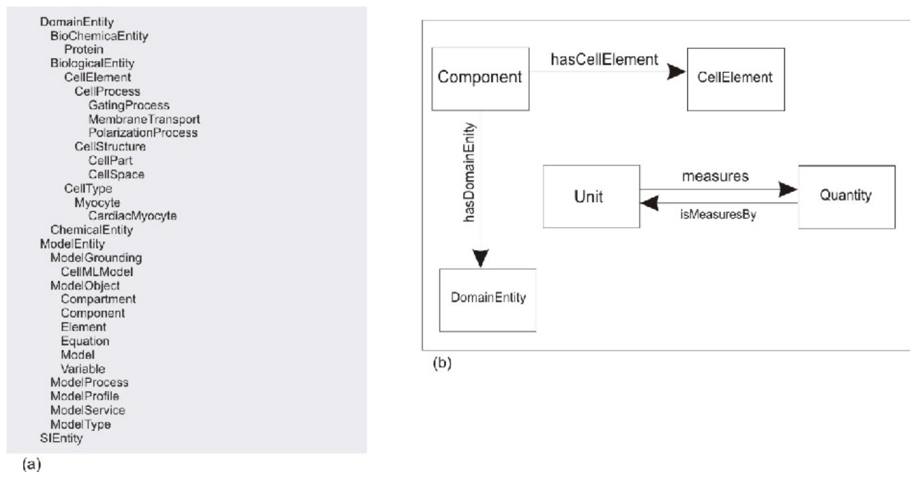


Fig. 1. Intermediate representation for the CelO ontology: (a) part of a classification concepts tree; (b) diagram of binary relationship of some concepts

In this paper we present only the *ModelEntity* subclasses. The complete CelO ontology and the performed tests are available in http://celo.mmc.ufjf.br.

The *ModelEntity* class and its subclasses (Fig. 2) define the used concepts in the biological model representation. The goal is to have a high level description, making reference to the CellML models for the simulation activity.

A challenging issue in the CelOWS architecture is to treat a model as a web service that has interfaces and can "be executed" (through simulation). The *ModelService* class provides a way of organizing the model vision as a service. Its structure is similar to OWL-S ontology [11], for semantic description of web service. An instance of the *ModelService* exists for each model and is associated to an instance of *ModelProfile, ModelGrounding* and *ModelProcess.*

ModelProfile gives information about the model: components association with some specific compartment of the cell and the biological entities associated to the model. *ModelGrounding* specifies the associated models, in case of a simulation. The *CellMLModel* class stores the URI of the associated CellML model. *ModelProcess* indicates how the model can be used, which are the input parameters (*ModelParameterIn*) and output ones (*ModelParameterOut*) that are associated to the interface model (*ModelInterface*). These parameters are directly associated to the model variables and can be used in the simulation or composition processes. *ModelType* is a generic class, used to characterize the biological problem of the model and its mathematical modeling.

ModelObject groups objects that compose a model and that are directly related to the underlying model. *Equation* names equations that implement the math model, expressed in MathML. *Variable* represents model variables. *ModelVariable* describes the variables roll in model equations and *ComponentVariable* is related to the variable of each model component. These variables can be used in the interface of the component (*InterfaceVariable*) or be local (*LocalVariable*). *DomainVariable* associates model variables with concepts of the *DomainEntity* class. *Model* represents the model itself, it can be atomic (only one component) or composed (two or more components). *Component* represents model components, that can be described in the proper model (*InternalComponent*) or can make reference to external ones (*ExternalComponent*). A model can be composed simultaneously by internal and external models.

The semantic rules are used to infer knowledge that is implicit in existing CellML models. The association between component variables and ontology individuals are defined and extracted from details in the CellML model. For example, the information may be extracted from the name of a variable and from predefined knowledge. For instance, we may have a rule that forbids the association of variables without dimension to chemical elements. Actually the rules can be also used to validate semantically the model in terms of consistency and completeness. A validation example is if the components group follows the anatomical hierarchy. Some rules of the CelO ontology, written in SWRL, are presented in Fig. 3:

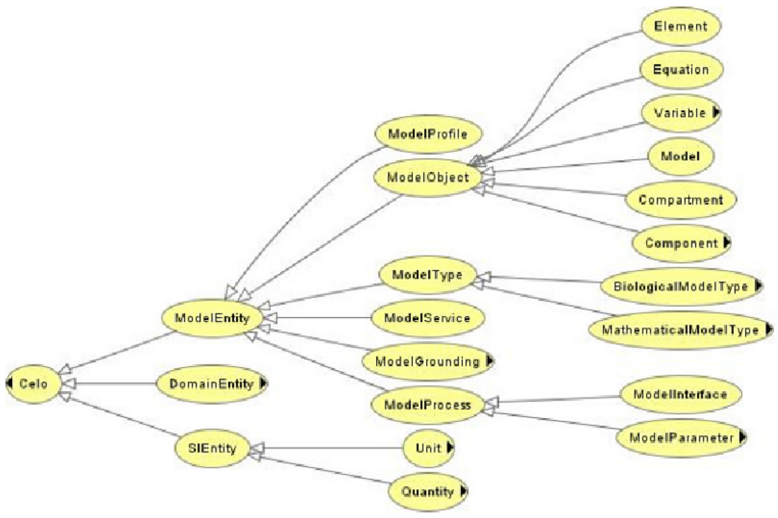
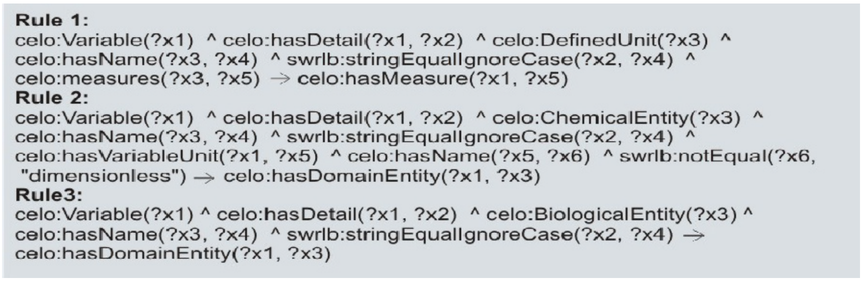


Fig. 2. Parcial representation of the CelO ontology in Protegé



Rule 1:
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:DefinedUnit(?x3) ^ celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) ^ celo:measures(?x3, ?x5) → celo:hasMeasure(?x1, ?x5)
Rule 2:
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:ChemicalEntity(?x3) ^ celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) ^ celo:hasVariableUnit(?x1, ?x5) ^ celo:hasName(?x5, ?x6) ^ swrlb:notEqual(?x6, "dimensionless") → celo:hasDomainEntity(?x1, ?x3)
Rule3:
celo:Variable(?x1) ^ celo:hasDetail(?x1, ?x2) ^ celo:BiologicalEntity(?x3) ^ celo:hasName(?x3, ?x4) ^ swrlb:stringEqualIgnoreCase(?x2, ?x4) → celo:hasDomainEntity(?x1, ?x3)

Fig. 3. Examples of the CelO ontology rules

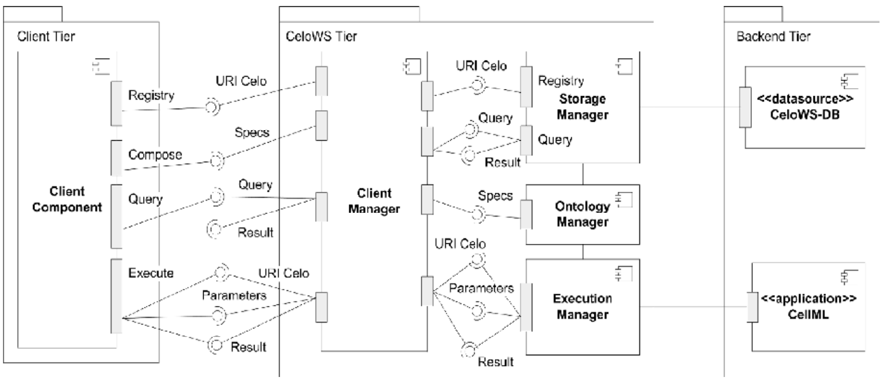


Fig. 4. Partial view of CelOWS framework architecture

There are different languages to represent the ontologies. Our research uses OWL , which is recommended by the W3C as the standard language used for web semantics projects. For inferences on individuals of the ontology the SWRL (Semantic Web Rule Language) was used [12]. SWRL makes possible the enrichment of the ontology with information not directly processed by the inference engine. This language was select because it is based on OWL.

4 CelOWS: An Architecture for e-Science Applications

The CelO ontology is the base for querying and composition of biological models in a service oriented architecture named CelOWS. This architecture uses the concepts of ontology repository and semantic web services. CelOWS aims to provide an infrastructure to register, research, recovery, compose and execute (simulate) biological models using ontologies. The combination of semantic description of components, in a standard format that allows their composition with other components, to models that can be simulated by existing tools, brings a great flexibility for the modeling processes in e-science projects.

CelOWS is implemented as a web service. This allows the repositories distribution and facilitates its use in scientific workflows and grid environments. Each model is also encapsulated in a web service, so it is able to be executed remotely (independently or composed with other components) or it can simply inform its localization where the local execution code can be found.

A general vision of the CelOWS architecture is presented in Fig. 4. The CelO URI represents the ontology model. The architecture considers three different tiers:

- *CelOWS*: Implemented as a web service, can be installed in different sites, allowing the distribution of the repositories, and is interface independent, as it facilitates the integration with existing tools.
- *Backend*: it is the services tier, used by the CelOWS to access the database (CelOWS-DB) and the execution tools to simulate CellML models.
- *Client*: implements the user interface and it can be developed in any language with access to web services.

The architecture offers four services for its users: a) *Registry*: From the URI given by the user, the model is then stored in a database.; b) *Compose*: The user gives XML files with the specification of the models that must be composed and the composition architecture (how the components will be connected among themselves). A new model is generated and stored in the database; c) *Query*: The user makes a SPARQL query [13] and receives, as a result, the models, components or variable that attends it; d) *Execute*: The user gives the URI model and the parameters to be used (from a previous query). From the given URI of the CellML model and a simulation tool is located, the computing is then executed and the results returned.

The CelOWS services are distributed in four layers: a) *Client Manager*: Responsible for all users interaction, implementing the Facade project pattern. Its purpose is to supply only one input/output point in the system, so customers do not have access to the internal structure of the CelOWS. b) *Storage Manager*: Responsible for the storage/recovery processes of the ontology in the database, and queries carried out by the user, encapsulating the access to the data base. c) *Ontology Manager*: Responsible for the inference on the models, as well as for providing an API (Application Program Interface) to access CelO ontology. d) *Execution Manager*: Responsible for the execution of the CellML model associated with the CelO model. The access to the simulation tools must be encapsulated, enabling the architecture to be independent of a specific tool.

The use of the complete CelOWS infrastructure depends on the development of end users applications, using graphical interfaces and encapsulating its services.

Prototype validation: To test and validate CelOWS functionalities a prototype was built, with the following services: register, query and models composition.

Eight CellML models were used for testing the Models Composition service. For example, the model "A Modification of the Hodgkin-Huxley Equations Applicable to Purkinje Fibre Action and Pace-Maker Potentials" is represented in Fig. 5 through a UML Components Diagram. The diagram presents the components and its connections through the interface parameters

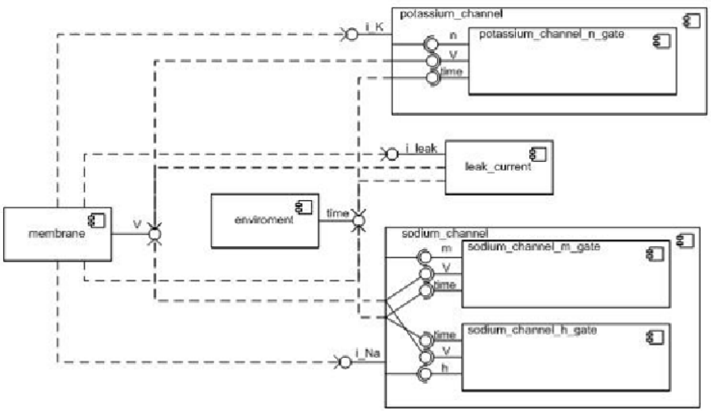


Fig. 5. Model used for testing the prototype

As it can be seen in the diagram, the model is composed of eight components: environment, membrane, potassium_channel_n_gate, potassium_channel, sodium_channel, leak_current, sodium_channel_m_gate, sodium_channel_h_gate. During the tests, the model was divided, creating eight CellML files (atomic models) and enabling eight CelO models. The original files, the membrane atomic model and the CelO model of the membrane are available in the site (http://celo.mmc.ufjf.br).

One of the specific goals of CelOWS is the composition of atomic models. The idea is to promote the reuse of existing components, through the composition of simpler models, producing more complex ones. The composition produces a new CellML model, through the copy of previously defined components and the automatic connection of these components.

The composition process uses a XML configuration file to indicate which models will be composed and the structure of the new model. The connection of the components is made through the semantic combination of the parameters of the first component with the parameters of the second one. It is established if both parameters measure the same quantities, if both are associated to the same chemical element and, finally, if both have the same name. A SPARQL query is used to find if the output parameters are compatible to the input of the other component. As the composition process is made through the copy of the existing models, it can be considered static (the modification of a used model in the composition process does not modify the composed component).

In order to validate the CellML model, generated from the composition process, the PCEnv program version 0.3, for models simulation, was used. Graph V x Times representing the potential action was generated from the original CellML "the Modification of the Hodgkin-Huxley Equations Applicable you the Purkinje Fibre Action and Pace-Maker Potentials" (Fig. 6) and with the file generated through the composition and connection of components (Fig. 7). These graphs (as others generated during the tests) are identical, proving the validation of the composition process.

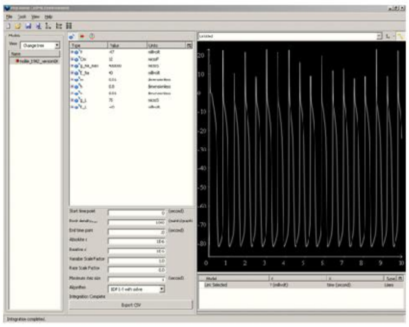


Fig. 6. Graph V x time generated with original CellML model

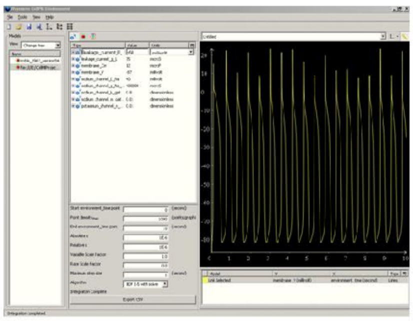
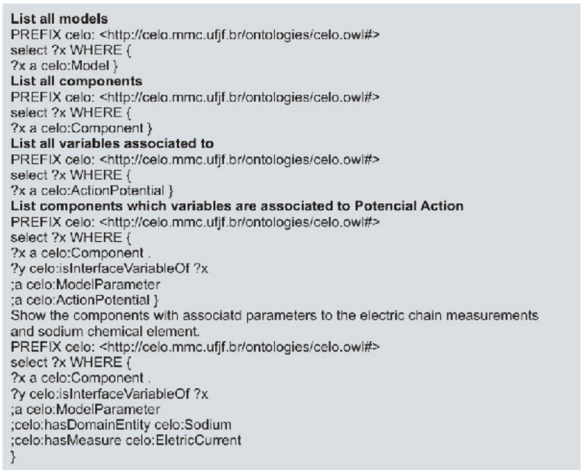


Fig. 7. Graph V x time generated with CellML model from a CelO model

To test and validate the semantic query functionalities in CelO models, some SPARQL queries (Fig. 8) were defined to locate variables or components that could be used in the composition process of a new model.

Implementation details: All the CelO ontology manipulation, as well as the described models in OWL, are in Protégé-OWL 3,4 (Editor and API) [14]. For inference the Pellet Reasoner [15] and the Jess Rule Engine [16] were adopted. For the development of the CelOWS we used the Eclipse, Java and PHP5 languages. For the ontology storage in a relational database and the SPARQL queries, the SOR (Scalable Ontology Repository) [17] was used.



List all models
```
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
?x a celo:Model }
```
List all components
```
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
?x a celo:Component }
```
List all variables associated to
```
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
?x a celo:ActionPotential }
```
List components which variables are associated to Potencial Action
```
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
?x a celo:Component .
?y celo:isInterfaceVariableOf ?x
;a celo:ModelParameter
;a celo:ActionPotential }
```
Show the components with associatd parameters to the electric chain measurements and sodium chemical element.
```
PREFIX celo: <http://celo.mmc.ufjf.br/ontologies/celo.owl#>
select ?x WHERE {
?x a celo:Component .
?y celo:isInterfaceVariableOf ?x
;a celo:ModelParameter
;celo:hasDomainEntity celo:Sodium
;celo:hasMeasure celo:EletricCurrent
}
```

Fig. 8. SPARQL queries in ontological repository

5 Related Work

MONET Project [18] investigates the area of mathematical web services, with implementation of a broker, definition of the description services language MSDL (Mathematical Service Description Language), many domain ontologies and the component InstanceStore [19]. The CelOWS architecture is similar to the Monet architecture even though Monet does not use semantic web services. Project GENSS (Grid-Enabled Numerical and Symbolic Services) [20] is an extension of Monet project. It deals with the combination of grid computing and mathematician web services using an open framework based in software agents. One of CelOWS implementation goal as a web service is to allow its use in workflow environments for services composition and its use in grid computing. In this way some of the GENSS project proposals could be applied to our research. An association of ontologies and web services to support the biological systems modelling is described in [21]. An ontology to represent the models in OWL is presented. OWL-S is used to specify the parameterization and semi-automatic composition of web services of the model execution. CelOWS architecture has similar goals, using, however, a more open domain ontology, the biological one. However, OWL is not used to represent the complete model, but to describe semantics of biological models in CellML

6 Concluding Remarks

The increasing volume and distribution of data and processes in Bioinformatics speed up the discovery of new biological information. To manage these data and processes in an automatic and scalable form, the use of scientific workflows is now essential. On the other hand, the biological models have different possible representations, such as conceptual, mathematical and computational. Although it is desirable the association of the workflows and modeling areas, the literature does not present specific proposals in this direction. We believe that this happens because generally the models are considered only in their "representation" aspect, while workflows deal with software components that can be "executed". Our research considers the approach of these areas using innovative technologies, based on well established standards. In the aspect of "representation" Cell Component Ontology - CelO is presented, to describe the semantics to the biological models. In the aspect of "process", the CelOWS architecture was developed to storage, query, reuse, compose and execute these models. CelOWS follows the service oriented architecture, being itself implemented as a web service. Some questions as scalability and performance have not been treated yet. Future work will focus on these questions, as well as the integration with other ontologies of the biological domain.

References

1. Garny, A., et al.: CellML and Associated Tools and Techniques. Elsevier Ireland Ltd. (2007)
2. Cell Markup Language, `http://www.cellml.org`
3. Bechhofer, S., et al.: OWL web ontology language 1.0 reference, `http://www.w3.org/TR/owl-ref/`
4. Nickerson, D., Hunter, P.: The Noble cardiac ventricular electrophysiology models in CellML. Prog. Biophys. Mol. Biol. 90(1-3), 346–359 (2006)
5. Luo, C., Rudy, Y.: A Dynamic Model of the Cardiac Ventricular Action Potential - Simulations of Ionic Currents and Concentration Changes. Circulation Research 74, 1071–1097 (1994)
6. Gruber, T.R.: Towards principles for the design of ontologies used for knowledge sharing. In: Guarino, N., Poli, R. (eds.) Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer, Dordrecht (1994)
7. Borst, W.N.: Construction of Engineering Ontologies. Phd Thesis (1997), `http://www.ub.utwente.nl/webdocs/inf/1/t0000004.pdf`
8. Barbosa, C., Santos, R., Amorim, R., et al.: A Transformation Tool for ODE based models. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3991, pp. 69–75. Springer, Heidelberg (2006)
9. Physiome CellML Environment, `http://www.cellml.org/downloads/pcenv`
10. Fernández-López, M.: Overview of Methodologies For Building Ontologies. In: Proceedings of the IJCAI 1999 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden (1999), `http://www.lsi.upc.edu/~bejar/aia/aia-web/4-fernandez.pdf`
11. Martin, D., et al.: OWL-S: Semantic Markup for Web Services. In: W3C Member Submission November 22 (2004), `http://www.w3.org/Submission/OWL-S/`
12. Horrocks, I., et al.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. In: W3C Member Submission, May 21 (2004), `http://www.w3.org/Submission/SWRL/`
13. SPARQL Query Language for RDF. W3C Recommendation, January 15 (2008), `http://www.w3.org/TR/rdf-sparql-query/`
14. Protégé-OWL – Ontology Editor for Semantic Web, `http://protege.stanford.edu/plugins/owl/`

15. Pellet: The Open Source OWL DL Reasoner, http://pellet.owldl.com
16. JESS, the Rule Engine for the Java Platform, http://herzberg.ca.sandia.gov
17. Lu, J., et al.: SOR: a practical system for ontology storage, reasoning and search. In: Proceedings of the 33rd international Conference on Very Large Data Bases, Vienna, Austria, September 23 - 27, 2007, Very Large Data Bases. VLDB Endowment, pp. 1402–1405 (2007)
18. MONET Consortium, http://monet.nag.co.uk
19. Bechhofer, S., Horrocks, I., Turi, D.: The OWL Instance Store: System Description. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS, vol. 3632, pp. 177–181. Springer, Heidelberg (2005)
20. GENSS, http://genss.cs.bath.ac.uk/project.htm
21. Sun, Z., Finkelstein, A., Ashimore, J.: Using Ontology with Semantic Web Services to Support Modeling in Systems Biology. In: International Workshop on Approaches and Architectures for Web Data Integration and Mining in Life Sciences (WebDIM4LS), Nancy, France (2007)

Benefits of Parallel I/O in Ab Initio Nuclear Physics Calculations

Nikhil Laghave[1], Masha Sosonkina[1], Pieter Maris[2], and James P. Vary[2]

[1] Ames Laboratory/DOE, Iowa State University, Ames, IA 50011, USA
{nikhill,masha}@scl.ameslab.gov
[2] Physics Department, Iowa State University, Ames, IA 50011, USA
{pmaris,jvary}@iastate.edu

Abstract. Many modern scientific applications rely on highly parallel calculations, which scale to 10's of thousands processors. However, most applications do not concentrate on parallelizing input/output operations. In particular, sequential I/O has been identified as a bottleneck for the highly scalable *MFDn (Many Fermion Dynamics for nuclear structure)* code performing *ab initio* nuclear structure calculations. In this paper, we develop interfaces and parallel I/O procedures to use a well-known parallel I/O library in MFDn. As a result, we gain efficient input/output of large datasets along with their portability and ease of use in the downstream processing.

Keywords: Parallel I/O, HDF5, MFDn.

1 Introduction

The direct solution of the quantum many-body problem transcends several areas of physics and chemistry. Our aim is to solve for the structure of light nuclei addressing the hurdles of a very strong nucleon-nucleon interaction, three-nucleon interactions, and complicated collective motion dynamics, by direct diagonalization of the nuclear many-body Hamiltonian in a harmonic oscillator single-particle basis.

The main tool used for the study of nuclear structure is the software package MFDn (Many Fermion Dynamics for nuclear structure) developed by Vary and his collaborators [1,2,3]. In MFDn, the nuclear Hamiltonian is evaluated in a large harmonic oscillator many-body basis and diagonalized by iterative techniques to obtain the low-lying eigenvalues and eigenvectors. The eigenvectors are then used to evaluate a suite of observables which can be compared to experimental quantities. One key feature of the quantum many-body calculations is the rapid increase of basis space dimension with the total number of particles and with the number of harmonic oscillator quanta allowed. The dimension of the basis space characterizes the size of the many-body matrix used to represent a nuclear many-body Hamiltonian. In general, the larger the basis set, the higher the accuracy of the energy estimation and other computable quantities one can obtain [4].

Despite the large dimension of the Hamiltonian matrix produced in nuclear structure calculations, it is sparse, meaning the matrix contains a large number of entries that are zero. The computational method used in MFDn to solve the matrix eigenvalues problem takes advantage of the sparsity structure of the Hamiltonian. The large dimension and the irregular sparsity structure of the Hamiltonian matrix pose a significant challenge to the algorithmic design, data structure specification, parallelization, and memory management strategies in MFDn for large-scale distributed-memory computer systems. Furthermore, the sparsity pattern of the matrix is not known in advance. It is determined efficiently during runtime by MFDn [2].

The original design of MFDn takes into account the standard parallel computing issues such as communication and load balancing [1]. The code has run successfully on as many as 30000 CPU's. The total number of processors on which MFDn is run is restricted to $\frac{n(n+1)}{2}$, where n are odd numbers. These, n processors are referred to as the "diagonal processors". MFDn code has evolved significantly [2,3], but one issue that has not been addressed till now is the I/O performance. In this paper, we describe the benefits of using parallel libraries for performing the I/O. Apart from performance issues, there are other features provided by the parallel libraries that make it more useful for MFDn.

1.1 Overview of MFDn

The MFDn [1] parallel code is used for large-scale nuclear structure calculations in the No-Core Shell Model (NCSM) formalism [5,6], which has been shown to be successful for up to 16-nucleon problems on present day computational resources. It is also used successfully in No-Core Full Configuration (NCFC) applications to a similar range of nuclei [4]. The MFDn code is charged to compute a few lowest ($\approx$ 15) converged solutions, that is, the wave functions and eigenvalues, to the many-nucleon Schrödinger equation:

$$H\,|\phi\rangle = E\,|\phi\rangle \tag{1}$$

The wave functions are expressed as a linear superposition of Slater Determinants in the harmonic oscillator basis. The limit on the retained Slater Determinants is defined by N_{max}, the total number of oscillator quanta above the lowest configuration. These wave functions are then used to calculate a set of observables. The matrix H in (1) is the Hamiltonian operator, which is typically solved using Lanczos diagonalization since H is symmetric and sparse. However, the Lanczos iterative process may be very expensive due to the large dimension of H with many off-diagonal elements. The number of Lanczos iterations also increases significantly to converge the energy levels above the ground state. For example, for the ^{16}O nucleus in the $N_{max} = 6$ basis space, the ground-state energy level requires only 35 Lanczos iterations, while 14 excited states need at least 500 Lanczos iterations to converge. Note that, in this case, the Hamiltonian H has the dimension of 26,483,625.

MFDn constructs the many-body basis space on the n diagonal processors, evaluates the Hamiltonian matrix elements in this basis on $\frac{n(n+1)}{2}$ processors using efficient algorithms [2], diagonalizes the Hamiltonian to obtain the lowest eigenvectors and eigenvalues, then post-processes the wave functions to obtain a suite of observables for comparison with experimental values. The diagonalization produces the wave functions distributed over n diagonal processors. These wave functions are then written to disk and read from disk in subsequent subroutines to verify numerical convergence of the diagonalization procedure and to calculate a suite of observables. Since the diagonal processors hold the wave functions, it is desirable that the I/O of the wave function file is done directly by the n diagonal processors. With heavier nuclei and larger N_{max} value, the size of the wave function file becomes very large and requires a substantial amount of I/O [4]. The dimension of the many-body basis for various nuclei and N_{max} values is tabulated below, along with the total number of processors selected, the total number of diagonal processors, and the wave function file size. Near term plans include matrices in the range of 10-20 billion dimension. The file size follows the growth in dimensions and thus provides scope for introducing parallel libraries to do this I/O. The wave function file is written and read once in a normal MFDn run. This file contains typically 15 eigenvectors of the size as indicated by the *Dimension* column in Table 1.

Table 1. Wave function dimensions and file sizes for a range of realistic test problems

Nucleus	N_{max}	Dimensions	CPU Count($\frac{n(n+1)}{2}$)	Diagonals(n)	File Size(GBytes)
^{12}C	6	32,598,920	190	19	1.82
^{6}He	14	155,710,094	4,950	99	8.70
^{12}C	8	594,496,743	8,646	131	33.22
^{16}O	8	996,878,170	12,090	155	55.70
^{14}F	8	1,990,061,078	27,730	235	111.20

1.2 Motivation for Using Parallel Libraries

I/O has been identified as a major bottleneck in parallel codes and there is a substantial interest in using parallel libraries that can make use of the underlying file system on parallel machines. Two widely used I/O libraries are the HDF5 library [7], and the NetCDF Library [8]. NetCDF is mostly used for array-oriented data whereas HDF5 is also used for storing raster images, complex scientific data and multidimensional arrays. Both libraries have a parallel implementation built on top of MPI-IO, supporting access to files in parallel. In addition to improved I/O efficiency, the use of these parallel I/O libraries has advantages such as portability, human readable files, and self-describing file formats.

In MFDn, the size of the basis space, and thus of the wave functions, becomes very large with heavier nuclei and larger N_{max}, as can be seen from Table 1. Construction of the Hamiltonian matrix and the diagonalization of the matrix using a Lanczos algorithm are the most time-consuming operations in MFDn; however, sequential I/O of the wave functions will become a bottleneck as the dimension of the basis space increases. After diagonalizing the Hamiltonian matrix, the wave functions are available at the diagonal processors. In the current version of MFDn, this I/O takes place sequentially with diagonal processors exchanging the data with the root processor and the root processor performing the actual I/O. We have modified this part of I/O in MFDn to make it parallel by having each diagonal processor write its portion of the data to a single file. The detailed implementation of this will be presented in section 2.2. Since the files are self describing and platform independent, they can be easily archived and used in the future by other groups.

Another advantage of using HDF5 is that data can be added anytime in the HDF5 file. In MFDn, the wave functions previously written into the file are read from the file and the correctness of the data is verified. Furthermore, certain properties like binding energy, total spin and isospin corresponding to each wave function are calculated. It is useful to store these properties along with the wave function, but in a raw binary file, this information cannot be appended to each wave function and had to be added at the end of the file. With HDF5, the space for these properties could be reserved and written later when these properties are calculated. In addition to the wave functions, it is essential to write related data such as a self contained description of the many-body basis space to disk in a portable format. This information can be further used to calculate certain observables outside of MFDn which involves post-processing of the wave functions. Thus, portability and self-describing information are desired properties since this post-processing can be done at a later time on another machine by other researchers. In addition to this, parallel I/O can also be useful for the input of the interaction files to MFDn, in particular the 3-body interaction files which are extremely large.

Node failures, segmentation faults and hardware maintenance interruptions are a stark reality in supercomputing. For large jobs running for many hours, a failure or exceeding wall clock time can result in non-normal job termination. In many cases this results in a major loss of resources. In such cases, it is imperative that large jobs have checkpointing enabled, so that a checkpoint file is created at regular intervals. In cases of abrupt node failures and machine outages, a checkpointed job could be restarted from the last checkpoint file. Portability of the checkpoint file is not very useful for MFDn since it is not feasible to move very large checkpoint files across machines for restarting MFDn jobs. Checkpointing for MFDn is under investigation using MPI-IO.

In MFDn, it may be valuable to save the sparsity structure of the matrix in a self describing format, to facilitate restarts of MFDn with different input data for the same nucleus. The use of parallel I/O libraries for large output files and its potential advantages in checkpointing serve as major motivations for using parallel I/O libraries over serial I/O. This paper focuses on performance improvements using self describing portable formats which we do not contemplate for checkpointing.

1.3 Wave Function File

In this section, the hierarchical structure of the wave function file created with HDF5 is described. To get a better understanding of the file structure, we briefly describe the file structure of an HDF5 file. A file is only a container for storing scientific data. The actual data is in fact stored in other objects inside the file. There are two primary objects in a file that contain the actual data, namely Groups and Datasets. Additional data like attributes and storage properties needed for data organization are also a part of the HDF5 file. Groups are similar to directories and serve as parent directories to datasets, attributes and other sub-groups. Datasets are the actual objects that contain the data of interest. Each dataset set contains the required data array and related meta-data. This meta-data is the data required for self description of files as well as data needed to maintain portability of files. Another very important object of an HDF5 file is dataspace. Dataspace in itself does not contain any data, but is a permanent part of dataset definition. Each dataset has an associated dataspace that contains the rank and dimensionality of the dataset. With this basic understanding of the file structure, we can now understand the wave function file of MFDn. Figure 1 shows the structure of the wave function file in MFDn where the lowest eigenvectors (≈ 15) are stored in different groups with each eigenvector written in parallel by all the diagonal processors.

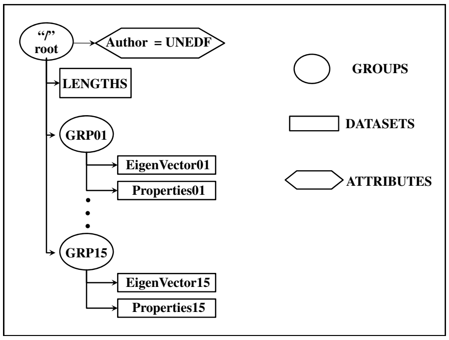


Fig. 1. Wave Function File

The rest of the paper is organized as follows. In section 2 we give an overview of HDF5 for MFDn. In this section, we discuss the differences between two modes of parallel I/O and situations in which these modes are useful. The integration of these parallel libraries with MFDn is also discussed in this section. In section 3, we explain the various experiments and I/O simulations that we have performed using HDF5. This is followed by the performance results in section 4. Finally, we conclude in section 5.

2 Using Parallel HDF5 in MFDn

Both parallel HDF5 and parallel NetCDF libraries provide a high level API for creating self-describing portable files, and both libraries support a wide range of operations. We use HDF5 for the experiments presented here since it appears to be sufficiently flexible and has a large user group. We do not investigate

NetCDF in detail. Parallel HDF5 has both an MPI-IO implementation and a POSIX implementation [9]. There are two modes of I/O that can be performed while doing parallel I/O, namely Independent I/O and Collective I/O.

Some parallel implementations work with the one file per processor approach in parallel I/O. This approach is the fastest but it is prohibitively expensive for programs running on large number of processors, since it requires expensive post-processing. Another method is to do I/O into a single shared file which can be written to and read by all the processors. This is the recommended way of performing parallel I/O and is supported both in NetCDF and HDF5. The parallel libraries provide a single file image to multiple processors, and multiple processors can do parallel I/O in the shared file. Each file is opened with a communicator that is passed as a parameter while creating the parallel file. Once created, a file handle is returned and all future accesses are performed using this handle. Once a file is opened by processes within the communicator, the file is then accessible to all the processors within the communicator. Thereafter, all objects within the file can be modified by these processors.

2.1 Performance with Collective and Independent I/O

Parallel I/O can be performed in two ways, collective I/O and independent I/O. Independent I/O means that each process can do the I/O independent of the other processes. Once a file is opened in parallel using a communicator, the file can then be accessed and modified independently by any of the processes in the communicator. In collective I/O however, all processes have to participate in I/O. Since all processes are required to participate, there is obvious inter-process communication required. However, in most cases, multiple I/O requests are interleaved as a single read/write operation, yielding very high speedups. In case of contiguous data layout in which each processor has a contiguous selection, if the selection of each processor is larger than the buffer size of the I/O agent, then independent I/O and collective I/O would both offer approximately the same performance. This occurs because in contiguous data, the data layout is already optimized and collective I/O cannot perform any more optimizations to improve I/O. Moreover, because of the inter-process communication in collective I/O, the performance of independent I/O is better compared to collective I/O since it does not have the overhead of inter-process communication. On the other hand, if each processor has a non-contiguous selection, then the interleaved access requests of different processes can be combined into a single contiguous I/O operation yielding a very high speedup with respect to independent access. That is, with independent I/O each processor has to perform several reads and writes in order to write the whole data leading to high cost of latency [10]. Apart from data layout, I/O performance is also affected by other factors such as caching, network bandwidth, latency, and the file system used.

HDF5 has another driver for parallel I/O besides MPI-IO. This driver is the MPI POSIX driver and is a "combination" of MPI-IO and posix I/O driver [9]. MPI is implemented for coordinating the actions of several processes and posix I/O is used to perform the actual I/O to the disk. This implementation does not support collective I/O mode. In independent I/O mode, MPI POSIX driver may perform better than MPI-IO driver [9], but in our experience the results obtained were identical with both posix and MPI-IO.

2.2 Integration with MFDn

Currently, the data is written to disk only by the root processor. The data is spread across the n diagonal processors and each diagonal processor sends the data to the root. The root writes its portion of the data and then receives the data from other processors, and then writes it to the disk. This process has obvious communication costs associated with it. Moreover the root processor takes all the load of writing to disk. We parallelized this task with the use of parallel I/O libraries. Since each diagonal processor has a part of the data that needs to be written to disk, that diagonal processor itself writes it to disk eliminating the cost of communication. The offset at which each diagonal processor writes the data is calculated during runtime in MFDn. Figure 2 shows the older sequential write pattern in MFDn and Figure 3 shows how parallel I/O is achieved. HDF5 uses the hyperslab model for doing I/O where hyperslabs are simple subsets of the dataspace. All the diagonal processors define their own non-overlapping hyperslab in the dataset and write the data into the file at specific offsets. To have a very simple and intuitive interface for writing files in HDF5 format, we have created two wrapper functions to read and write the data to file. The interface is straightforward and minimal HDF5 parameters are required while calling the routines for writing and reading the data. Separate modules for HDF5, MPI-IO, and raw binary modes of I/O will be created and, depending on the user choice, an appropriate I/O model will be used.

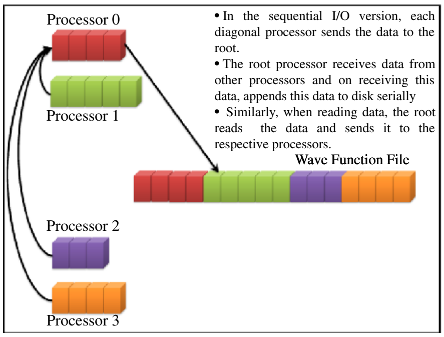

Processor 0
Processor 1
Processor 2
Processor 3
Wave Function File
• In the sequential I/O version, each diagonal processor sends the data to the root.
• The root processor receives data from other processors and on receiving this data, appends this data to disk serially
• Similarly, when reading data, the root reads the data and sends it to the respective processors.

Fig. 2. Sequential I/O

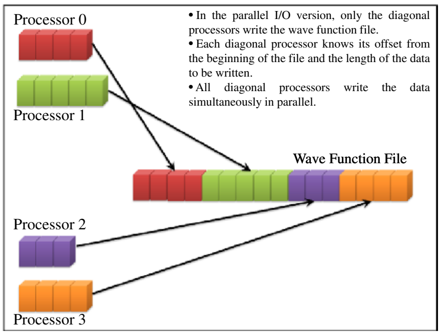

Processor 0
Processor 1
Processor 2
Processor 3
Wave Function File
• In the parallel I/O version, only the diagonal processors write the wave function file.
• Each diagonal processor knows its offset from the beginning of the file and the length of the data to be written.
• All diagonal processors write the data simultaneously in parallel.

Fig. 3. Collective I/O

3 HDF5 Performance Tests with MFDn

A normal run of MFDn requires $\frac{n(n+1)}{2}$ processors. After diagonalization of the Hamiltonian matrix, the wave functions are available on the n diagonal processors, and written to disk; the other processors are idle during the actual I/O of the wave functions. In order to test the I/O performance, we have created a suite of test programs to simulate the wave function I/O phase of MFDn on the n diagonal processors. In addition, we have a set of programs that can be used to generate the required input files. Once the raw binary and HDF5 files are available, the I/O performance is measured by a set of programs to simulate posix I/O and HDF5 I/O.

4 Performance Results

In this section, we report the I/O performance using the HDF5 library, as well as the sequential one processor I/O using our I/O test programs. The testbed for our experiments was the super computing Franklin cluster at NERSC. Franklin is a distributed-memory parallel system with 38,640 processor cores available for scientific applications. It has 9,660 compute nodes and each consists of a 2.6 GHz quad-core AMD Opteron processor with a theoretical peak performance of 9.2 GFlop/sec. Each compute node has 8 GBytes of memory. The parallel file system on Franklin is the Lustre file system. Figures 4-6 show the performance of parallel HDF5 for different file sizes. The processors on the X-axis represent the n diagonal processors. For same file sizes, the actual MFDn code would run on

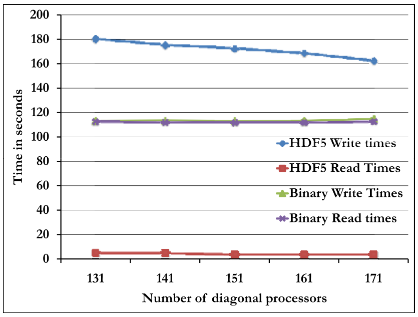


Fig. 4. I/O Performance for 33 GB File

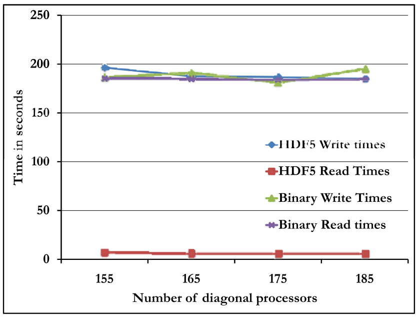


Fig. 5. I/O Performance for 55 GB File

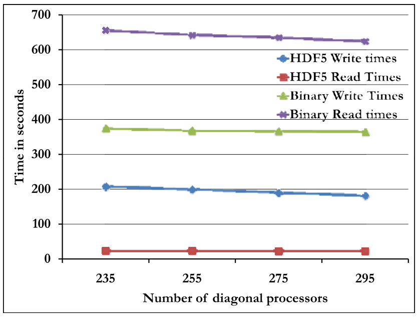


Fig. 6. I/O Performance for 111 GB File

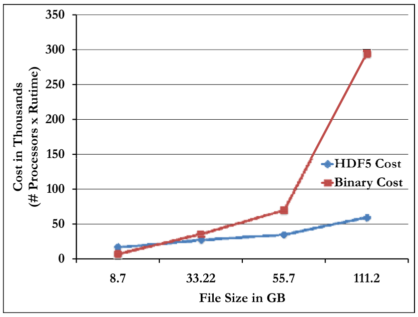


Fig. 7. Cost of using Parallel I/O

$\frac{n(n+1)}{2}$ processors for several hours. It is not feasible to run MFDn code for hours on large scale machines for observing I/O performance. Hence we demonstrate the results using the I/O test programs that we developed.

As can be seen in the plots, the HDF5 write operations are much slower as compared to read operations. The Lustre file system has write locks wherein each writer process acquires a lock on the file and this locking infrastructure effectively prevents simultaneous parallel writes. The parallel write operations are serialized due to the file locking resulting in poor write performance. On parallel file systems such as PVFS, where there are no write locks, the performance of parallel writes can be expected to be much better. Figure 7 shows the cost of using parallel I/O. We define cost as the product of the total number of processors and the total time for I/O. For small files (below 20 GB), parallel HDF5 I/O is more costly than the current sequential binary I/O implementation, but once the file size increases above 20 GB, parallel HDF5 becomes more cost-effective than sequential binary I/O. For the largest file sizes considered here, 111 GB, the difference between parallel HDF5 and sequential binary is approximately a factor of 5 in favor of HDF5. This means that despite locking infrastructure and the slow HDF5 write operations, the cost of using HDF5 is lesser compared to raw binary I/O for large datasets. An output file of about 20 - 100 GB is typical of our large runs where I/O is a significant factor.

We have done all experiments using independent I/O mode since, for contiguous data, independent mode is faster than collective mode. There are several HDF5 users [11], but most of them use HDF5 for complex scientific data in collective I/O mode. According to our knowledge, this is the first real-world application using HDF5 for contiguous data in independent I/O mode.

The I/O performance of any parallel library is also affected by the parallel file system on the clusters. With the Lustre file system, multiple processes cannot write to the same file at the same time. Each time a process has to write, it acquires a lock on the file. So when a new process has to write anything, it sends a request for lock revocation. Only when the lock has been released by the previous writer, can the new process acquire a lock on the file. These are all costly operations and this is the reason for the high write times we can see in the performance charts above. As a work around of this defect in the Lustre and MPI-IO interface, MPI-IO hints can be passed to HDF5 via the MPI_INFO parameter. The *cb_nodes* parameter indicates the total number of writers to be used. By passing a value of “1” to this parameter, we can redirect all the I/O through a single processor. This does not address the root problem but tries to reduce the costly operations of acquiring and revoking locks since all data is written by a single writer. With parallel reads however, there are no locks and this is not the problem, leading to much faster reads. In spite of the slow parallel writes, we intend to have a common I/O approach for all parallel file systems. Computer scientists are currently working to address the problem of MPI-IO and Lustre interface, so we expect better I/O performance from newer version of MPI-IO and Lustre interface.

5 Conclusion and Future Work

We have identified sequential I/O as a bottleneck in the MFDn code performing nuclear structure calculations. We have successfully implemented parallel I/O for the optimization of the MFDn code. The results obtained encourage the use of parallel I/O libraries for sufficiently large datasets. Even for file systems that have a locking infrastructure for parallel write operations, the cost of using parallel I/O is less compared to sequential I/O for sufficiently large datasets. Our contribution is to show how parallel I/O libraries can be of great value to scientific applications dealing with large datasets. We have investigated the difference between collective and independent I/O and situations in which they are effective. With these results in hand, we plan to implement parallel I/O for other large datasets used by MFDn.

Acknowledgments

The work was supported in part by Iowa State University under the contract DE-AC02-07CH11358 with the U.S. Department of Energy, by the U.S. Department of Energy under the grants DE-FC02-07ER41457 (UNEDF SciDAC-2) and DE-FG02-87ER40371 (Division of Nuclear Physics), and by NERSC.

References

1. Vary, J.: The Many-Fermion Dynamics Shell-Model Code. Iowa State University (unpublished) (1992)
2. Sternberg, P., Ng, E.G., Yang, C., Maris, P., Vary, J.P., Sosonkina, M., Le, H.V.: Accelerating configuration interaction calculations for nuclear structure. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Conference on High Performance Networking and Computing, Austin, Texas, November 15 - 21, 2008, pp. 1–12. IEEE Press, Piscataway (2008), http://doi.acm.org/10.1145/1413370.1413386
3. Sosonkina, M., Sharda, A., Negoita, A., Vary, J.P.: Integration of Ab Initio Nuclear Physics Calculations with Optimization Techniques. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 833–842. Springer, Heidelberg (2008)
4. Maris, P., Vary, J.P., Shirokov, A.M.: Ab initio no-core full configuration calculations of light nuclei. Phys. Rev. C 79, 014308 (2009) arXiv:0808.3420 [nucl-th]
5. Navratil, P., Vary, J., Barrett, B.: Properties of ^{12}C in the ab-initio Nuclear Shell Model. Phys. Rev. Lett. 84, 5728 (2000)
6. Navratil, P., Vary, J., Barrett, B.: Large-basis ab-initio No-core Shell Model and its application to ^{12}C. Physical Review C62, 054311 (2000)
7. The HDF Group, http://www.hdfgroup.org
8. The NetCDF Homepage, http://www.unidata.ucar.edu/software/netcdf/
9. Yang, M., Koziol, Q.: Parallel HDF5 Hints. NCSA HDF group
10. Chilan, C.M., Yang, M., Cheng, A., Arber, L.: Parallel I/O Performance Study With HDF5, A Scientific Data Package. In: TeraGrid 2006: Advancing Scientific Discovery, June 12-15 (2006)
11. HDF5 USERS, http://www.hdfgroup.org/HDF5/users5.html

A Population-Based Approach for Diversified Protein Loop Structure Sampling

Yaohang Li

Department of Computer Science
North Carolina A&T State University
Greensboro, NC 27411
yaohang@ncat.edu

Abstract. Protein loop structure modeling is regarded as a mini protein folding problem with significant scientific importance. Efficiently sampling the loop conformation space is a key step to computationally obtain accurate loop structure models. Due to the large size of the conformation space and the complication of the scoring functions describing protein energy, it is difficult to obtain broad, diverse coverage of the loop conformations with low energy (score). In this article, we present a new population-based approach to sample the backbone conformations of protein loops. The main advantage of the population-based approaches is that various selection schemes can be applied to enforce the conformations in a population to satisfy certain constraints. In our sampling approach, conformations are generated in the dihedral angles (ϕ, ψ)-space and the Differential Evolution (DE) method is employed to implement dihedral angle crossover for generating new conformations. A diversity selection scheme is applied to achieve diversified sampling. Using a narrowing gap selection scheme, decoys satisfying loop closure condition are obtained by gradually eliminating conformations with large terminal gaps in a population. Our computational results on modeling long loop targets have shown diverse and broad coverage of the loop conformation space, which leads to consistently reaching the native-like decoys in the sampling process.

1 Introduction

Protein loop structure modeling is important in structural biology for its wide applications, including determining the surface loop regions in homology modeling [1], defining segments in NMR spectroscopy experiments [2], designing antibody [3], and modeling ion channel [4]. The value of computer-generated protein loop models in biological research and practice relies critically on their accuracy. Protein loop structure modeling can be considered as a mini version of the *ab initio* protein folding problem. Despite their short length, protein loops exhibit greater structural flexibility than strands and helices and have few contacts with the remainder of the protein, which make it more difficult to predict than the geometrically regular β-strands and α-helices [5]. Currently, development of high-resolution computational approaches that can reliably produce accurate protein loop models, particularly in long loop targets, remains an unsolved problem. The main difficulties include the large protein loop conformation space as well as the complicated landscape of the scoring functions describing the loop energy.

Similar to *ab initio* protein folding, the rationale of *ab initio* protein loop structure modeling is to optimize a protein loop energy function to discover the native-like conformations [7]. Typically, in protein modeling, the physics-based energy functions yield a rugged, funnel-like energy landscape, which can easily trap the optimization process and is extremely difficult to search. Several approaches, such as replacing the van der Waals potential with a soft-sphere potential [8], switching to a statistics-based term [9], etc., have been developed to produce scoring functions with "softened" energy landscape to facilitate the search process. However, such scoring functions also come with insensitivity and potentially inaccuracy, i.e., a conformation with the absolutely lowest score may not be a native-like conformation while a conformation with a relatively higher score may in fact be a more reasonable structure than the one with a lower score. Therefore, it is well-known that an optimization method seeking the very global minimum of a scoring function is usually not effective in finding the true native conformation. Instead, a sampling approach that can efficiently explore the low score regions in the scoring function landscape is more desirable [10].

For very short protein loop targets, one may be able to traverse the discretized dihedral angles (ϕ, ψ)-space to completely sample all possible conformations. However, for longer protein loops, the size of the conformation space grows exponentially where complete sampling becomes infeasible. Markov Chain Monte Carlo [6, 11] and genetic algorithms [12] have been applied to sample the loop conformation space to discover feasible structures with low scores (energy). The existing problems in these sampling methods include oversampling – the same conformations are repeatedly generated as well as undersampling – some conformations with low scores are not reachable during the sampling procedure. Oversampling will lead to wasted computational efforts while more seriously, undersampling may miss good, native-like conformations.

In this article, we present a population-based sampling algorithm to achieve broad exploration of protein loop backbone conformation space. We use the backbone dihedral angles (ϕ,ψ) array as a reduced representation of a loop conformation. A modified Differential Evolution (DE) scheme [13] is used to crossover dihedral angles of selected conformations in an old population to generate new conformations in (ϕ,ψ)-space. A diversity selection scheme is developed to filter conformations in a population similar to those already generated during the sampling procedure, which favors the sampling process to explore undiscovered conformations with low scores and thus reduces the chance of repeatedly generating decoys with similar structures. By gradually eliminating the conformations in a population not satisfying the loop closure condition, our narrowing gap selection scheme can also lead to decoys with loop closure satisfaction. We verify our sampling approach by applying it to the long targets provided in Jacobson's protein loop benchmark [14].

The remainder of the article is organized as follows. Sections 2 and 3 describe the general protein loop structure modeling procedure and our population-based sampling method, respectively. Section 4 shows our computational results on the 12-residue loop benchmark targets. Section 5 summarizes our conclusions and future research directions.

2 General Protein Loop Structure Modeling Paradigm

The *ab initio* protein loop structure modeling procedure [15, 16, 17] typically involves the phases of sampling, filtering, clustering, and refining, although some additional steps may be employed in different programs. Figure 1 shows a conceptual illustration of these phases.

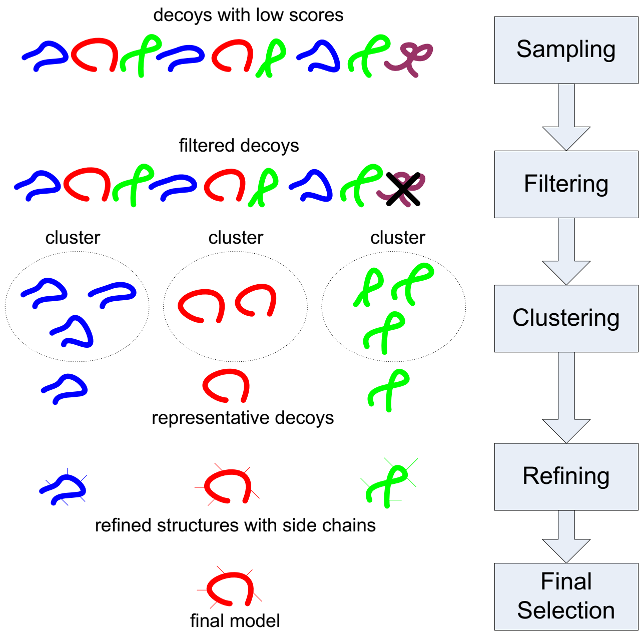


Fig. 1. Typical steps in high resolution *ab initio* protein loop structure modeling

In the sampling phase, the loop conformation space is explored and decoys with low scores are produced. In order to reduce the degree of freedom, usually only loop backbone with reduced representation are used in this phase with simplified, smooth scoring functions. Afterward, the infeasible, bad decoys will be eliminated in the filtering phase. Then, in the clustering phase, decoys with similar structures will be grouped into a cluster and representative decoys for each cluster will be selected. Next, in the refining phase, side chains are added and complicated all-atom energy functions are used to locally optimize the representative decoys. Finally, the refined representative decoy with the lowest energy will be selected as the predicted model.

Broadly sampling the loop conformation space to generate low-score decoys with diverse structures in the sampling phase is critical in successfully predicting high-resolution protein loop models. This is due to the fact that if a native-like decoy is not reachable in loop conformation sampling, it is unlikely to obtain a high-resolution model close to the native structure in the refining phase. For the population-based sampling approach described in this article, we only consider the modeling computation in the sampling phase.

3 Population-Based Diversified Sampling Approach

We develop a population-based sampling approach which intends to diversely sample the loop conformation space. Initially, a population with N conformations, $C_1, \ldots, C_N$, is randomly generated. Each loop structure conformation C_i with n residues is represented by a vector $(\theta_1, \ldots, \theta_{2n})$, which represents the backbone dihedral angles of $(\phi_1, \psi_1, \ldots, \phi_n, \psi_n)$. The dihedral angles of ω_i are kept constants at their average value of 180°. A statistical distance-based atom pair-wise scoring function is used as the sampling scoring function [18]. When scoring function evaluation or structure comparison is needed, the dihedral angles representation of C_i is converted to the backbone atom representation. We adopt the Differential Evolution (DE) [13] approach to produce new conformations for the next population. A diversity selection scheme is designed to achieve diversified sampling and a narrowing gap selection scheme is used to guarantee loop closure.

3.1 Differential Evolution for Conformation Crossover

DE [13] is a powerful computational method for continuous function optimization, which has demonstrated its effectiveness on several hard optimization problems with complicated objective functions [22]. In our loop sampling approach, DE is used to crossover old conformations to produce new ones in continuous dihedral angles space. For each loop conformation C_i, a mutant vector V_i is formed by

$$V_i = C_{r1} + F(C_{r2} - C_{r3}), \tag{1}$$

where $r1$, $r2$, and $r3$ are mutually distinct, uniformly distributed integer random numbers in the interval $[1, N]$ and $F > 0$ is a tunable amplification control constant as described in [4]. Then, a new conformation $C_i'(\theta_1', \ldots, \theta_{2n}')$ is generated by the crossover operation on V_i and C_i:

$$\theta_j' = \begin{cases} v_j & j = \langle s \rangle_{2n}, \langle s+1 \rangle_{2n}, \ldots, \langle s+L-1 \rangle_{2n} \\ \theta_j & otherwise \end{cases} \tag{2}$$

where $\langle \cdot \rangle_{2n}$ denotes the modulo operation with modulus $2n$, s is a randomly generated integer from the interval $[0, 2n\text{-}1]$, L is an integer drawn from $[0, 2n\text{-}1]$ with probability $\Pr(L = k) = (\mathrm{CR})^L$, and $\mathrm{CR} \in [0, 1)$ is the crossover probability. Practical advice suggests that CR = 0.9 and F = 0.8 are favorable choices in the DE scheme [13], which is also adopted in our program. Our slight modification to the DE scheme is to always keep θ_i in the ranged of $[-\pi,\pi]$.

3.2 Diversity Selection Scheme

The diversity selection scheme encodes the capability of enforcing the conformations in a population to satisfy the diversified sampling requirement. Our diversity selection scheme is based on the similarity of a conformation s to a given set of generated decoys $D = \{d_1, \ldots, d_k\}$, which is measured by

$$S(s) = \min_i dist(s, d_i) \tag{3}$$

where *dist*(.) is a distance function measuring the structural difference.

In our implementation, we keep track of the already generated decoys d_1, …, d_k by recording their Cα atoms in an array. To reduce computation time of evaluating loop structure similarity, instead of calculating the Root Mean Square Deviation (RMSD) of all backbone atoms, we use the Cα RMSD between conformations in a population and the generated decoys as the distance function. Then, in diversity selection, all conformations in the current population are sorted according to their similarity to the generated decoys and the top μ% of the candidates are eliminated, where μ is a tunable constant.

3.3 Narrowing Gap Selection Scheme for Loop Closure

The so-called loop closure problem is defined as follows: given the N- and C-terminals, find a loop backbone conformation of a certain length that can bridge the ends seamlessly [19]. Inverse kinematics [23] is a common method to solve the loop closure problem. Unfortunately, inverse kinematics has difficulty to be applied to our population-based sampling approach because crossing over the dihedral angles of two or more conformations satisfying the loop closure condition does not automatically guarantee loop closure in the new conformation.

In our population-based sampling approach, we develop a narrowing gap selection scheme to produce decoys satisfying the loop closure condition. We fix the position of the N-terminal, produce the loop based on the dihedral angle values in loop conformation C_i, and then calculate the gap distance, $G(C_i)$, from the C-terminal in the generated loop to the target C-terminal. $G(C_i)$ is then used to measure the loop closure gap. To produce loops closely approximating the loop closure condition, in the gap selection scheme, we eliminate conformations C_i where $G(C_i) > \delta$. Here δ is a variable, which specifies the acceptable gap between the predicted C-terminal and the target C-terminal. At the beginning, δ is initialized to a large value to allow aggressive loop conformation sampling. The value of δ is decreasing in every iteration toward a small value so as to gradually eliminate conformations with gaps larger than δ and eventually lead to conformations approximately satisfying the loop closure condition. When the final conformation with the lowest score is selected to output as a decoy, the C-terminal gap can continue to be reduced by slightly adjusting the dihedral angles of ϕ_i, ψ_i, and ω_i in the loop structure.

3.4 Algorithm Description

By putting every piece of the puzzle together, the descriptive pseudo code of the population-based diversified sampling algorithm is described as follows. The algorithm can be repeatedly executed to produce multiple loop decoy structures.

```
Initialize $N$ conformations, $C_1$, …, $C_N$, randomly and initialize $\delta$
Repeat {
    Generate $M$ new conformations, $C_1'$, …, $C_M'$, based on the previous
       population's $N$ conformations using DE
    Run diversity selection scheme to eliminate conformations close to
       the already generated decoys stored in the decoy array
    Run gap selection scheme to eliminate conformations that $G(C_i'>\delta)$
    Evaluate the remaining $C_1'$, …, $C_M'$ use scoring function $f(.)$
    Replace $C_1$, …, $C_N$ with top N conformations in $C_1$, …, $C_N$ and the re-
       maining $C_1'$, …, $C_M'$
    Reduce $\delta$
} Until convergence or reaching the expected iteration number
Produce the decoy in the current population with the lowest score
If there is serious steric clash or large loop closure gap
    Discard this decoy
Else {
    Save $C_\alpha$ atoms to the generated decoy array
    Minimize the terminal gap of by slightly adjusting the dihedral an-
       gles of $•_i$, $•_i$, and $•_i$
    Output the loop decoy}
```

4 Computational Results

We applied our methods to the long loop benchmark targets specified in [14], including 17 12-residue, 35 11-residue, and 49 10-residue loops. Due to space restrictions, we can only report a fraction of our results in this article. Therefore, we use our computational results on 12-residue loop targets to illustrate the effectiveness of our population-based diversified sampling scheme. Our computations on the other targets actually yield similar results.

We use the path length of the Minimum Spanning Tree (MST) [20] based on the pair-wised RMSD matrix of the generated decoys to measure sampling diversity. Table 1 shows the comparison of the MST path lengths of the 1,000 decoys generated by our population-based loop conformation sampling algorithm with and without the diversity selection scheme. It is important to notice that the diversity selection scheme plays an important role in the sampling process, which leads to significantly larger MST path length in all 12-residue loop targets when the diversity selection scheme is employed. In other words, the decoys generated using the diversity selection scheme are more structurally different from each other than those without using the diversity selection scheme. This indicates that the sampling process with the diversity selection scheme has a broader coverage of the loop structure conformation space and leads to decoys with more diversified representation of structures.

The diversified sampling of the loop conformation space directly improves the chance of generating decoys with close structure to the native one. Table 1 also compares the best decoys with the smallest backbone RMSD to the corresponding native structure generated with and without using the diversity selection scheme in 12-residue targets. One can find that in all loop targets except for 5nul(54:65), the population-based sampling process with the diversity selection scheme can consistently reach decoys with backbone RMSD less than 2A, which is within the experimental X-ray crystallization resolution. In contrast, sampling without the diversity selection scheme cannot reach decoys with RMSD less than 2A in 5 out of the 17 targets. Moreover, there is averagely 0.37A RMSD shift in the best decoys of the 12-residue targets when the diversity selection scheme is used.

Table 1. MST path length of the pair-wise RMSD matrix and the best decoy with the smallest backbone RMSD of the 1,000 decoys generated in our population-based sampling with and without the diversity selection scheme in 12-residue loop targets

			With Diversity Selection Scheme		Without Diversity Selection Scheme	
Protein	Start Res.	End Res.	MST Path Length (A)	RMSD (A) of the Best Decoy	MST Path Length (A)	RMSD (A) of the Best Decoy
1ixh	160	171	1436	1.519	1297	2.786
1cex	40	51	1294	1.780	1201	2.265
5pti	36	47	1389	1.610	1295	1.844
1rge	57	68	1178	1.005	1132	1.403
1arb	74	85	1211	1.376	1115	1.500
7rsa	13	24	1343	1.509	1207	2.102
1xyz	813	824	1281	1.510	1151	1.687
1cyo	32	43	1368	1.341	1243	1.444
1akz	181	192	1370	1.197	1255	1.912
153l	98	109	1399	1.791	1308	2.245
1bkf	9	20	1296	1.102	1206	1.443
1dad	204	215	1382	1.423	1270	1.717
1dim	213	224	1262	1.109	1190	1.624
1kuh	90	101	1371	1.214	1258	1.188
2ayh	21	32	1392	1.678	1245	1.540
351c	15	26	1383	1.914	1271	1.612
5nul	54	65	1341	2.141	1235	3.218
Average			1335	1.483	1228	1.855
Standard Deviation			71	0.312	57	0.529

Our further structural analysis shows that the native 5nul loop (54:65) interacts with a flavin mononucleotide ligand. The distance-based scoring function used in our sampling program makes no assumption on any ligands. This explains why in 5nul(54:65) no decoys with RMSD under 2A are generated in our sampling approach even when the diversity selection scheme is used.

Due to the broad structure representations in the generated decoys, sampling with the diversity selection scheme will also lead to diversified clusters and representative decoys in the clustering phase of protein loop structure modeling. Figures 2 and 3 show the representative decoys in clustering the 1,000 decoys in loop target 1akz(181:192) using sampling with and without the diversity selection scheme, respectively. We use a simple agglomerative clustering algorithm [21] with 2.0A cutoff. One representative decoy is selected for each cluster. For the 1,000 decoys generated without using the diversity selection scheme, 9 clusters are produced. For each representative one of the 9 clusters, a similar structure can be found in the representative decoys from the 19 clusters generated by sampling using the diversity scheme. However, several representative decoys exhibiting significantly different structures found in sampling using the diversity

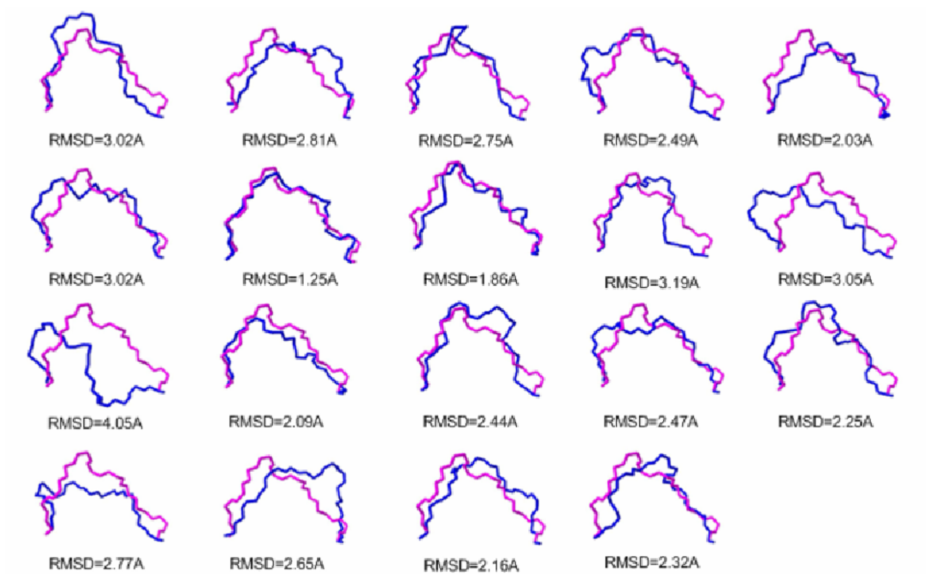


Fig. 2. Representative decoys in clustering the 1,000 decoys generated with diversity selection scheme in loop target 1akz(181:192). (purple – native conformation, blue – decoy conformation)

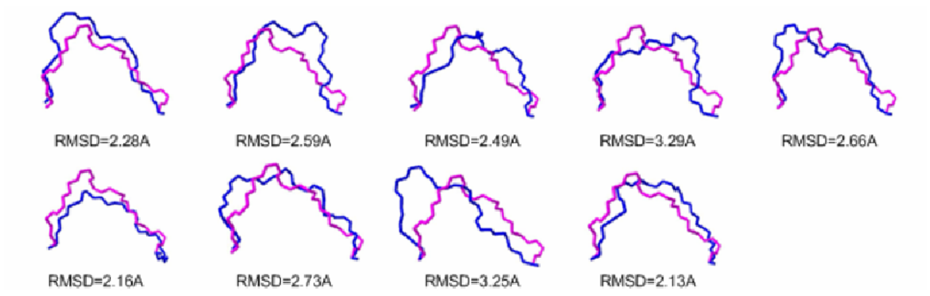


Fig. 3. Representative decoys in clustering the 1,000 decoys generated without diversity selection scheme in loop target 1akz(181:192). (purple – native conformation, blue – decoy conformation)

selection scheme are not presented in these 9 clusters generated by sampling without diversity selection scheme, including a native-like one with 1.25A RMSD.

A minor disadvantage of the diversified sampling method is that it may also increase the production of "bad" decoys. This is due to the fact that the diversity selection scheme will increase the chance of discovering not only the "good", native-like conformations but also the "bad", far-deviated ones yielding low scores. As an example depicted in Figure 4 showing the RMSD distribution of the 1,000 decoys in loop target 1rge(57:68), sampling with the diversity selection scheme leads to larger population of decoys with RMSD less than 2.0A as well as those with RMSD higher than 3.0A than sampling without diversity selection scheme. This problem can be relatively easy to address in the filtering phase of loop structure modeling – when a

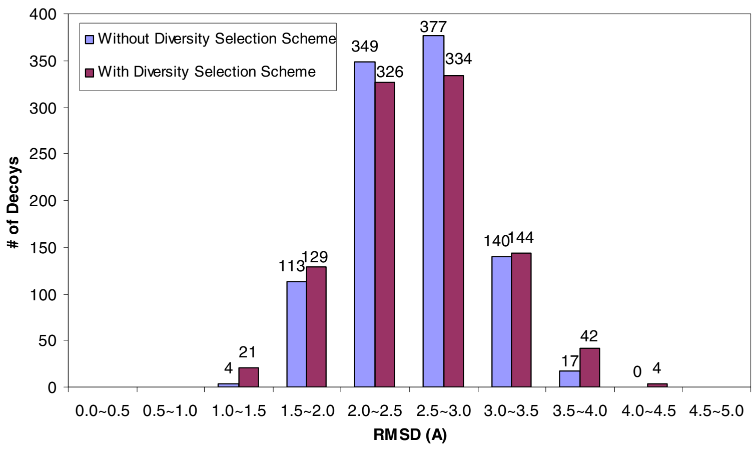


Fig. 4. RMSD distribution of the 1,000 decoys generated in sampling with the diversity selection scheme and without the diversity selection scheme in loop target 1rge(57:68)

high-resolution, all-atom scoring function is employed, the "bad", far-deviated decoys can usually be identified and then eliminated.

5 Conclusions and Future Research Directions

In this article, we present a population-based sampling algorithm for diversified sampling protein loop backbone conformations. A diversity selection scheme is designed to diversify predicted decoys and a narrowing gap selection scheme is used to achieve loop closure condition satisfaction. Our computational results on 12-residue protein loop benchmark targets have shown diversified decoy structure distributions and improved chance of reaching native-like conformations.

It is important to notice that our approach only targets the backbone sampling phase in *ab initio* protein loop structure modeling and the decoy generation time is usually less than a minute. As a result, our decoys have relatively lower quality compared to the all-atom modeling method such as PLOP [15], which typically takes days to deliver a model. In the future, we are interested in studying how diversified backbone sampling can benefit all-atom simulation in high-resolution loop modeling.

Acknowledgement

The work is partially supported by NSF under grant number CCF-0829382.

References

1. Bruccoleri, R.E.: Ab initio loop modeling and its application to homology modeling. Methods in Molecular Biology 143, 247–264 (2000)

2. Dmitriev, O.Y., Fillingame, R.H.: The rigid connecting loop stabilizes hairpin folding of the two helices of the ATP synthase subunit c. Protein Science 16(10), 2118–2122 (2007)
3. Martin, A.C., Cheetham, J.C., Rees, A.R.: Modeling antibody hypervariable loops: a combined algorithm. PNAS 86(23), 9268–9272 (1989)
4. Tasneem, A., Iyer, L.M., Jakobsson, E., Aravind, L.: Identification of the prokaryotic ligand-gated ion channels and their implications for the mechanisms and origins of animal Cys-loop ion channels. Genome Biol. 6(1), R4 (2005)
5. Monnigmann, M., Floudas, C.A.: Protein loop structure prediction with flexible stem geometries. Proteins: Structure, Function, and Bioinformatics 61(4), 748–762 (2005)
6. Cui, M., Mezei, M., Osman, R.: Prediction of protein loop structures using a local move Monte Carlo approach and a grid based force field. Protein Engineering Design and Selection 21(12), 729–735 (2008)
7. Anfinsen, C.B.: Principles that Govern the Folding of Protein Chains. Science 181, 223–230 (1973)
8. Zhang, H., Lai, L., Han, Y., Tang, Y.: A Fast and Efficient Program for Modeling Protein Loops. Biopolymers 41, 61–72 (1997)
9. van Vlijmen, H.W.T., Karplus, M.: PDB-based protein loop prediction: parameters for selection and methods for optimization. J. Mol. Biol. 289, 1469–1490 (1999)
10. Li, Y., Bordner, A.J., Tian, Y., Tao, X., Gorin, A.: Extensive Exploration of the Conformational Space Improves Rosetta Results for Short Protein Domains. In: 7th International Conference on Computational Systems Bioinformatics (2008)
11. Liu, Z., Mao, F., Li, W., Han, Y., Lai, L.: Calculation of protein surface loops using Monte Carlo simulated annealing simulation. Journal of Molecular Modeling 6(1), 1–8 (2000)
12. McGarrah, D.B., Judson, R.S.: Analysis of the genetic algorithm method of molecular conformation determination. Journal of Computational Chemistry 14, 1385–1395 (1993)
13. Storn, R., Price, K.: Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. J. of Global Optimization 11(4), 341–359 (1997)
14. Jacobson, M.P., Pincus, D.L., Rapp, C.S., Day, T.J.F., Honig, B., Shaw, D.E., Friesner, R.A.: A hierarchical approach to all-atom protein loop prediction. Proteins 55(2), 351–367 (2004)
15. Zhu, K., Pincus, D.L., Zhao, S., Friesner, R.A.: Long loop prediction using the protein local optimization program. Proteins 65, 438–452 (2006)
16. Rohl, C.A., Strauss, C.E., Chivian, D., Baker, D.: Modeling structurally variable regions in homologous proteins with rosetta. Proteins 55, 656–677 (2004)
17. de Bakker, P.I.W., Depristo, M.A., Burke, D.F., Blundell, T.L.: Ab initio construction of polypeptide fragments. Proteins 51, 21–40 (2002)
18. Rojnuckarin, A., Subramaniam, S.: Knowledge-based interaction potentials for proteins. Proteins: Structure, Function, and Genetics 36, 54–67 (1999)
19. Canutescu, A.A., Dunbrack, R.L.: Cyclic Coordinate Descent: A Robotics Algorithm for Protein Loop Closure. Protein Science 12, 963–972 (2003)
20. Xu, Y., Olman, V., Xu, D.: Minimum Spanning Trees for Gene Expression Data Clustering. Genome Informatics 12, 24–33 (2001)
21. Lance, G.N., Williams, W.T.: A general theory of classificatory sorting strategies. The Computer Journal 9, 373–380 (1967)
22. Price, K., Storn, R., Lampinen, J.: Differential Evolution – A practical approach to global optimization. Springer, Heidelberg (2005)
23. Kolodny, R., Guibas, L., Levitt, M., Koehl, P.: Inverse Kinematics in Biology: the Protein Loop Closure Problem. International Journal of Robotics Research 24(3), 151–163 (2005)

Parameter Space Exploration Using Scientific Workflows

David Abramson, Blair Bethwaite, Colin Enticott,
Slavisa Garic, and Tom Peachey

Faculty of Information Technology,
Monash University,
Clayton, 3800, Victoria, Australia
{david.abramson,blair.bethwaite,colin.enticott,
slavisa.garic,tom.peachey}@infotech.monash.edu.au

Abstract. In recent years there has been interest in performing parameter space exploration across "scientific workflows", however, many existing workflow tools are not well suited to this. In this paper we augment existing systems with a small set of special "actors" that implement the parameter estimation logic. Specifically, we discuss a set of new Kepler actors that support both complete and partial sweeps based on experimental design techniques. When combined with a novel parallel execution mechanism, we are able to execute parallel sweeps and searches across workflows that run on distributed "Grid" infrastructure. We illustrate our new system with a case study in cardiac cell modelling.

Keywords: distributed workflows, parameter exploration, Kepler.

1 Introduction

In recent years there has been a great deal of interest in "scientific workflows" [10]. These allow scientists to specify large computational experiments involving a range of different activities, such as data integration, modelling and analysis, and visualization. Activities can be composed, often using a graphical programming environment, so that the output of one stage can be passed as input to the next, forming a pipeline of arbitrary complexity. Scientific workflow engines manage the execution across a range of distributed resources, and leverage Grid computing middleware and approaches [1][2][3]. For example, it is possible to extract data from a scientific instrument, pass it through some analysis software running on a high performance cluster, store the results in a distributed data repository, and then visualize it on a large display wall. Scientific workflows have been used to great effect in a number of different disciplines including Computational Chemistry [7], Ecology[8] and Bio informatics [9].

Importantly, many workflow engines double as programming environments for the Grid. Whilst there are no standard ways of doing this, a number of engines effectively expose the Grid middleware APIs. For example, Kepler [5] exposes a variety of middleware layers, from Globus through to ad-hoc interfaces like SSH. Other engines, such as Triana [19] and Taverna [9] allow users to invoke services as Web Services, but provide no explicit support for Grid middleware.

Prior to the wide adoption of workflow engines, we developed a family of tools, called Nimrod, for performing parameter sweeps with computational models. Nimrod supports the execution of a specific type of workflows in which a single computation is performed multiple times to allow exploration of some design space. Nimrod includes tools that perform a complete parameter sweep across all possible combinations (Nimrod/G) [4], or search using non-linear optimization algorithms (Nimrod/O) [20] or experimental design techniques (Nimrod/E) [21]. Importantly, the degree of parallelism, can be varied at run time, as the Nimrod scheduler places tasks on the available resources then. Nimrod had been applied to a wide range of disciplines from public health policy to quantum chemistry [22].

However, Nimrod was not designed to execute arbitrary workflows of the type discussed above. Thus, it is difficult to run sweeps *over* workflows, and workflows *containing* sweeps. Likewise, most workflow systems do not support the parallel execution of tasks that are supported in Nimrod, and are not well suited to parameter sweeps and searches.

In this paper we discuss how we have added parameter sweeps and searches to existing workflow tools. We focus on the techniques that allow a range of scenarios to be explored by adding a few simple components to an existing workflow. Specifically, we discuss the design of a new family of Kepler "actors" that support sweeps across parameter ranges. The resulting system is extremely flexible, and allows the creation of decision support systems of arbitrary complexity.

The paper begins with a discussion of parameter sweeps and the techniques that are commonly used, illustrating the ideas through the Nimrod family of tools. We then discuss workflow engines, and in particular, we describe Kepler. We then introduce a new family of actors, and show how these can be combined with existing workflows. Finally, we demonstrate the applicability of the ideas using a case study in cardiac modelling.

2 Parameter Sweeps

2.1 Full Parameter Sweeps

Many current scientific and engineering problems can be formulated as computational models with a great deal of accuracy. Changing the inputs allows a user to explore a range of design scenarios, giving a picture of how the system behaves. This is typically performed as a sweep over the input parameters. Although the model may be computationally expensive, parallel execution of the jobs can dramatically speed up the execution and allow very large systems to be studied.

In this context, it is useful to think of a given computational model as a function that accepts a set of input parameters and produces a set of outputs. Input parameters are typically simple types, such as integers, floats and text strings. A full parameter sweep takes all combinations of the parameters, and allows exploration of the entire design space within some finite resolution. Such parameter sweeps have been used very effectively, for example in environmental modelling [23], bioscience [24], engineering [20] and chemistry [25].

2.2 Partial Parameter Sweeps

Although the favoured method for exploring the parameter space of a model is a full parameter sweep, this may be impractical where there are many parameters, especially if the model is computationally intensive. Experimental design techniques may enable meaningful results to be obtained from a suitably chosen subset of jobs.

The most widely used method of experimental design is the fractional factorial design [11]. Suppose the experiment has input parameters (called factors in the literature) *A, B, C, …* and produces a numerical output *f(A,B,C…)*. The underlying model is that *f* is the sum of several terms, a constant term f_0, then the "main effect" terms $f_1(A)$, $f_2(B)$, …, each dependent on only one factor, then "two-factor interaction effects" terms, $f_{1,2}(A,B)$, $f_{1,3}(A,C)$, $f_{2,3}(B,C)$, …, each dependent on two factors, then "three-factor interactions", and so on. Long experience, with both physical and computer-based experiments, suggests that the higher-order terms, interactions of three factors and more, are usually negligible. In that case, only the lower-order terms need to be estimated, so fewer jobs are needed to obtain these estimates and hence obtain reasonable approximations to the output. Significant savings are possible; with 20 parameters, each at two levels, the number of jobs required is reduced from over a million to 512.

2.3 The Current Nimrod Tool Set

Over the past 15 years we have constructed a tool set called Nimrod [4], that automates both of the parameter sweep techniques discussed above. One of the Nimrod tools, Nimrod/G, supports complete parameter sweeps It operates either as a tool, or a middleware layer in its own right. If the former, then it is usually operated from a Web portal, and this allows a user to create a plan file, set up a testbed, manage certificates etc., and organize input and output files from an experiment. Nimrod/G, however, can also serve as a job management system for other software, including the other members of the Nimrod family. This structure is shown in Fig. 1.

The Nimrod/E tool, on the other hand, automates the design of fractional factorial experiments. Here, the user specifies the factors and which interactions can be ignored. Then, one component produces an efficient design generating the parameter values for the resulting jobs in a form suitable for Nimrod/G. When all jobs are complete, another component produces analyses of the results for each output value of interest. It creates graphs showing the relative size of the various main effects and interactions. The Daniel Plot [13], Fig. 6, plots the effects so that the negligible ones, with values just experimental error, will form the central straight line, significant effects produce points that deviate from this line. The Lenth Plot [14], Fig. 5, shows effects in order of absolute size and horizontal lines giving significance levels. Points outside the outer lines are most probably significant. The tool also estimates results of a full parameter sweep, which may then be used by visualization software.

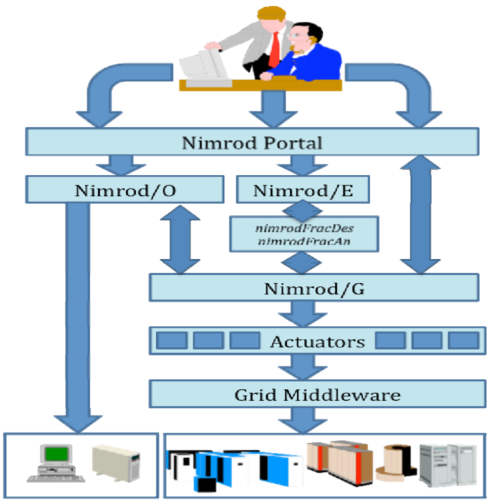


Fig. 1. Nimrod tool chain

3 Workflow Engines and Kepler

There has been considerable interest in scientific workflow engines over the past few years, in particular, in Grid workflow engines. Workflows allow a user to build arbitrary computations from a set of connected components, or *actors*. Actors may represent computations, but also facilitate access to distributed databases and scientific instruments. In most systems, data moves along the edges that connect the actors as *tokens*, and can thus be streamed from instruments and databases, through a range of computational processes. Grid workflows allow these actors to be executed on distributed resources, and launched in a variety of ways, facilitating virtual applications that span multiple organizations, data sources and computers.

To date, many scientific workflow tools have been built [6][9][10]. Grid workflow systems allow general applications to be constructed, with examples ranging from ecology to medical imaging. In this project we focus on Kepler [5][6][7][8], which allows a user to weave a "virtual" application from a set of otherwise distinct components, and its workflow engine orchestrates their execution in a controlled and repeatable manner. Kepler builds upon Ptolemy II [11], a Java-based software framework with a graphical user interface called Vergil. Ptolemy II is used for the modelling, simulation, and design of concurrent, real-time embedded systems. The focus is on assembly of concurrent components. Kepler inherits a number of orchestration mechanisms from Ptolemy, providing an extensive range of execution mechanisms. These are controlled through devices called "directors".

In spite of their significant power, Kepler, and many other current workflow systems, do not support dynamic parallel execution of a workflow and its components. This means that users must explicitly code a workflow to cause it to execute elements in parallel – either by replicating the workflow statically, or adding looping constructs that scatter and gather threads. Both of these techniques significantly complicate the workflow and obscure the underlying business logic. In another recent paper, we have shown how to augment Kepler with a Tagged Data Flow Architecture Director (TDA) [18]. This new system, called Nimrod/K, extends Kepler by providing powerful mechanisms for exposing and managing parallelism in the workflows. This provides an ideal platform for using workflows for parameter sweeps. Unlike the current Nimrod tool chain, Nimrod/K makes it possible to run sweeps over workflows, and workflows that contain sweeps. It leverages Kepler's power in building complex workflows, and Nimrod's ability to execute sweeps over grid resources. In the next section we show how we have created some specific new Kepler actors that facilitate parameter sweeps. When combined with Nimrod/K's parallel execution mechanisms, we have a powerful new tool for parameter sweeps.

4 New Kepler Actors

4.1 Actor Design

The Nimrod/G Actor. As discussed in section 2.3, the current Nimrod tools provide parameter sweep tools to complement Nimrod/G's Grid execution and meta-scheduling capabilities. One of these tools performs a parameter sweep that creates a Grid job for each of the parameter combinations in the parameter space. Each of these parameter combinations is represented in Kepler using datasets, called "tokens". We have created a *ParameterSweep* actor, that uses the same parameter syntax as the Nimrod/G implementation, to generate parameter combinations in a workflow. This actor works with the current Ptolemy directors, however, when used in combination with the Tagged Dataflow mechanism in Nimrod/K, parallel execution of the different parameter combinations allows the workflow to execute efficiently on the Grid.

The new *ParameterSweep* actor is quite general. For example, multiple actors can be chained together to create a sub parameter sweep inside a larger parameter sweep. Although it is possible to create such a sweep with a single *ParameterSweep* actor, staging them allows for preliminary calculations of shared data to be performed by the parent sweep prior to the start of the sub-sweep. Also, chaining them together enables a more dynamic parameter range options, for example, if the sub sweep is calculated from the parameters given by the parent sweep. This environment produces sweeps that meet the parameter sweep requirements of workflows.

Nimrod/E Actors. The workflow for a typical Nimrod/E experiment is shown in Fig. 2. The user enters details of the parameters to be varied and the effects to be estimated, into the *FractionalFactorialDesigner* actor. This designs the experiment and specifies the jobs required, passing these job specifications (as workflow tokens) to the actor(s) that perform the model execution. If the model is a simple formula then the model execution may be performed by a Kepler numerical actor. More complex modelling may require execution of an external model via the Nimrod/K framework.

When all jobs are complete, downstream processing can begin. There may be several numerical aspects of the model output of interest, so at this stage the dataflow may branch. Fig. 2 shows the case where there are two outputs requiring analysis.

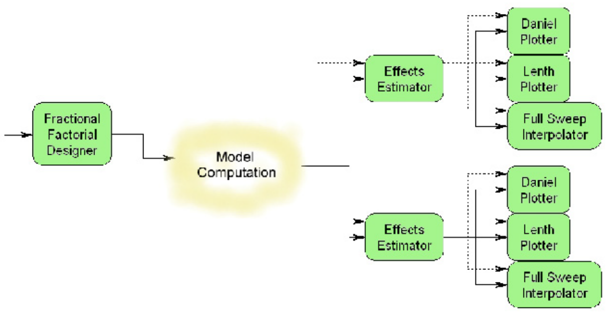


Fig. 2. Nimrod/E experiment

One output from each model is needed to estimate the effects for that output. Tabulation of the effects may be sufficient, or these estimates may be used as inputs for a Daniel plot, Lenth plot and/or computation of the full sweep results.

4.2 Implementation

Nimrod/G. The `ParameterSweep` actor is built on the PtolemyII source class that provides an output port and a trigger. The output port sends a RecordToken containing the value of each parameter for a single parameter combination. The actor has a property that contains the parameter sweep string identical to the syntax of the parameter sweep commands in the Nimrod plan file syntax. To implement the token tagging functions, the `ParameterSweep` actor is built on the classes and functions provided by the Nimrod/K TDA director. These classes have no functional affect when used with other Ptolemy directors. Under the Nimrod/K director each parameter combination is executed in parallel. This is further discussed in [18].

Nimrod/E. The actors for Nimrod/E were built using the same principles as the *`ParameterSweep`* actor. Because *`nimrodFracDes`* is computationally intensive there is a clear advantage in retaining the C code for this function. Further, the code is quite complex, so that re-writing in Java within the Kepler framework would be a large undertaking. Consequently, we decided to use the existing C code, using a Java wrapper to produce a design actor. The information that it produces as a plan file is redirected as a tag to the output token for downstream processing. This actor is named the *`FractionalFactorialDesigner`* in Fig. 2.

For the analysis section most computation is relatively light. Further, the graphs produced would benefit from a re-coding in Java, using the Kepler graphing functionality. When there are a large number of factors, the plots produced may display estimates for many effects and be quite cluttered; Kepler graphs provide zooming to allow the user to explore the fine detail. On the other hand, the production of the analysis matrix does require some of the complex computation used in *`nimrodFracDes`*. Accordingly, we decided to recode the analysis section within the Kepler framework. The *`EffectEstimator`*, *`LenthPlotter`*, *`Daniel-Plotter`* and *`FullSweepInterpolator`* actors were added to perform the analysis discussed in Section 2.3.

5 Case Study

In this section we illustrate the new actors we have discussed to date, using a case study previously performed with the Nimrod/G and Nimrod/E tools. The case study allows us to demonstrate the utility of the new approach and its expressive power in Kepler.

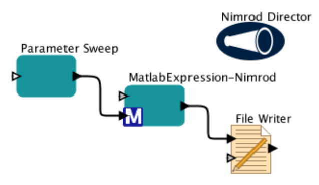


Fig. 3. Complete parameter sweep

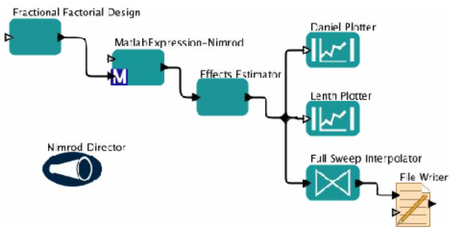


Fig. 4. Experimental Design

Mathematical models of the heart show considerable promise for understanding the underlying mechanisms and for clinical diagnosis and treatment [15][12]. The model chosen [16] concerns excitation and contraction in rabbit ventricular muscle cells. Intracellular flows of calcium, sodium and magnesium ions were modelled as a system of ordinary differential equations using Matlab. Earlier experiments with this model are reported in [17]. Fig. 3 shows the Kepler workflow used to perform a parameter sweep over the nine factors, *A* to *J.,* This used two values for each factor, producing an experiment of 512 jobs, and the values are generated using the `ParameterSweep` Actor.

A second experiment used the experimental design functionality to further investigate the response function. The `FractionalFactorialDesign` actor in the workflow shown in Fig. 4 was used to produce a design that would estimate the main effects and two-way interactions of these factors. This required only 128 jobs.

Fig. 5 shows the Lenth plot from the experiment. Points for the effects of all nine inputs lie outside the confidence lines and hence all inputs make a significant

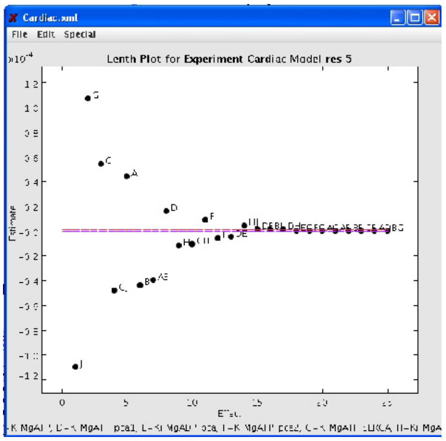


Fig. 5. Lenth plot

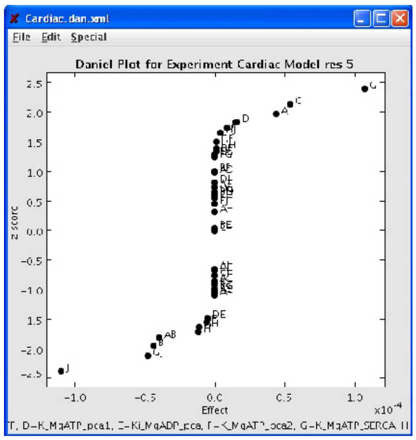


Fig. 6. Daniel plot

contribution. Of the interactions, only, *AB, DE, GH, GJ* and *HJ* are definitely significant, the other interactions are indistinguishable from noise. This reveals the structure of the computational model. The final output is the sum of four terms, the first affected only by *A* and *B*, the second by *C*, the third by *D, E* and *F* and the last by *G, H* and *J*, explaining why many interactions have no effect. However the lack of significant interactions between *D* and *F,* and between *E* and *F*, requires further investigation. Fig. 6 shows the same results on a Daniel plot. Here the deviation of points from the vertical line gives a measure of the significance of the effects plotted.

Fig. 7 shows the parallelism in the complete sweep when executed under the Nimrod/K Director. Our cluster was saturated at just over 100 concurrent executions. The system takes some time to build up the parallelism, and this is evident in the initial ramp up phase. This is caused by the overheads in starting remote computations using the Globus framework.

This case study demonstrates that Kepler provides a natural and easy mechanism for specifying the use of parameter sweep techniques over existing workflows. We have used a fairly simple computation model in this paper to clearly show the techniques, but this does not really demonstrate the significant potential of the approach.

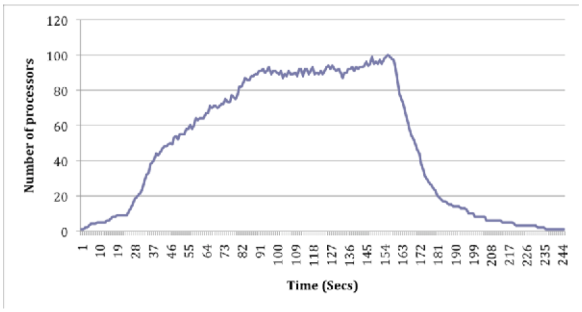


Fig. 7. Parallelism in complete sweep

All of the machinery developed to date is capable of sweeping across parameter combinations regardless of the complexity of the computational steps of the workflow. Thus, much more complex pre-existing workflows can be modified using our new actors to perform large-scale complex parameter sweep experiments. We have recently begun work on modifying a complex computational chemistry workflow by adding the actors discussed in this paper.

6 Conclusions

In this paper we have discussed the design and implementation of some new actors that facilitate parameter sweeps in scientific workflows. The solution builds on the existing execution frameworks in workflow systems, and we have demonstrated its applicability in Kepler. The simple case study shows how easy it is to create and execute a sweep over an existing scientific computation.

Our system leverages a separate addition to Kepler, called Nimrod/K, which supports parallel execution of the different instances. By combining the new Actors, and the new TDA Director, we can execute each of the parameter combinations in parallel. This makes it possible to compute complex design experiments quickly if there are sufficient resources available.

Acknowledgements

The Nimrod/K project is supported by Australian Research Council ARC Discovery grant. We are grateful to our colleague, Ilkay Altintas, from the San Diego Supercomputing Center for her assistance, and Anushka Michailova and Saleh Amirriazi, from the UC San Diego, for their work on the cardiac model.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
2. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Int'l J. of Supercomputer Applications 11(2), 115–128 (1997)
3. The Globus Toolkit, `http://www-unix.globus.org/toolkit/`
4. Abramson, D., Giddy, J., Kotler, L.: High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid. In: Int'l. Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico (2000)
5. Kepler Project, `http://kepler-project.org`
6. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. In: Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows (2005)
7. Altintas, I., Birnbaum, A., Baldridge, K., Sudholt, W., Miller, M., Amoreira, C., Potier, Y., Ludaescher, B.: A Framework for the Design and Reuse of Grid Workflows. In: Herrero, P., Pérez, M.S., Robles, V. (eds.) SAG 2004. LNCS, vol. 3458, pp. 120–133. Springer, Heidelberg (2005)
8. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., Kepler, M.S.: Towards a Grid-Enabled System for Scientific Workflows. In: the Workflow in Grid Systems Workshop in GGF10 - The 10th Global Grid Forum, Berlin (2004)
9. Taverna Project, `http://taverna.sourceforge.net`
10. Yu, J., Buyya, R.: A Taxonomy of Workflow Management Systems for Grid Computing. Technical Report GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, Univ. of Melbourne (2005)
11. Box, G.E., Hunter, W.G., Hunter, J.S.: Statistics for Experimenters. Wiley, Chichester (1978)
12. Amirriazi, S., Chang, S., Peachey, T., Abramson, D., Michailova, A.: Optimizing Cardiac Excitation-Metabolic Model By Using Parallel Grid Computing. In: Biophysics 2008, Long Beach, California (2008)
13. Daniel, C.: Applications of Statistics to Industrial Experimentation. Wiley, Chichester (1976)
14. Lenth, R.V.: Quick and Easy Analysis of Unreplicated Factorials. Technometrics 31, 469–473 (1989)

15. Hunter, P.J., Kohl, P., Noble, D.: Integrative Models of the Heart: Achievements and Limitations. Philosophical Transactions of the Royal Society of London: Mathematical, Physical and Engineering Sciences, 2001, 10.1098/rsta.2001.0816 (2001)
16. Michailova, A., Lorentz, W., McCulloch, A.: Modelling Transmural Heterogeneity of KATP current in Rabbit Ventricular Myocytes. AJP-Cell Physiology, 293, C542–C557 (2007)
17. Amirriazi, A., Chang, S., Peachey, T., Abramson, D., Michailova, A.: Optimizing Cardiac Excitation-Metabolic Model By Using Parallel Grid Computing. In: Biophysics 2008, Long Beach, California (February 2008)
18. Abramson, D., Enticott, C., Altinas, I.: Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows. In: IEEE Supercomputing 2008, Austin, Texas (November 2008)
19. Taylor, I., Shields, M., Wang, I.: Resource Management of Triana P2P Services. In: Grid Resource Management. Kluwer, Netherlands (2003)
20. Abramson, D., Lewis, A., Peachey, T., Fletcher, C.: An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics. In: SuperComputing 2001, Denver (November 2001)
21. Peachey, T.C., Diamond, N.T., Abramson, D.A., Sudholt, W., Michailova, A., Amirriazi, S.: Fractional Factorial Design for Parameter Sweep Experiments using Nimrod/E. Journal of Scientific Programming 16(2,3) (2008)
22. Message Lab list of current and former e-Science projects, `http://www.messagelab.monash.edu.au/EScienceApplications`
23. Lynch, A.H., Abramson, D., Beringer, K.J., Uotila, P.: Influence of savanna fire on Australian monsoon season precipitation and circulation as simulated using a distributed computing environment. Geophys. Res. Lett. 34, L20801 (2007)
24. Sher, A., Abramson, D., Enticott, C., Garic, S., Gavaghan, D., Noble, D., Noble, P., Peachey, T.: Incorporating local Ca2+ dynamics into single cell ventricular models using Nimrod/O. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 66–75. Springer, Heidelberg (2008)
25. Sudholt, W., Baldridge, K., Abramson, D., Enticott, C., Garic, S.: Parameter Scan of an Effective Group Difference Pseudopotential Using Grid Computing. New Generation Computing 22, 125–135 (2004)

Interactive Parallel Analysis on the ALICE Grid with the PROOF Framework

Marco Meoni

Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland
European Organization for Nuclear Research, 1211 Genève, Switzerland
marco.meoni@epfl.ch
http://people.epfl.ch/marco.meoni

Abstract. The ALICE experiment at CERN LHC is intensively using a PROOF cluster for fast analysis and reconstruction. The current system (CAF - CERN Analysis Facility) consists of 120 CPU cores and about 45 TB of local space. PROOF enables interactive parallel processing of data distributed on clusters of computers or multi-core machines. Subsets of selected data are automatically staged onto CAF from the Grid storage systems. However, a cluster of the size of CAF can only hold a fraction of the yearly 3 PB data accumulated by ALICE. The impracticability to store and process such data volume in one single computing centre leads to the need to extend the concept of PROOF to the Grid paradigm.

Keywords: ALICE, Grid, PROOF, interactive parallel distributed processing, data movement, task colocation, resource availability.

1 Introduction

The ALICE [1] experiment at the CERN Large Hadron Collider (LHC) will accumulate data at unprecedented speed and volume. The yearly estimate is 1.5 PB of raw data from the experimental setup and additional 1.5 PB of reconstructed data and Monte-Carlo simulations. The data management and processing is done through the ALICE Environment [2] (AliEn) middleware on the Worldwide LHC Grid [3] (WLCG [4]), encompassing hundreds of computing centres with many thousands of CPUs and PB scale disk and mass storage systems. A set of unique challenges for the reconstruction and analysis software and the ways the physicists perform data analysis is offered by the data volume and distributed computing environment.

One of the most important aspects of data analysis is the speed with which it can be carried out. Fast feedback on the collected data and publication of results is essential for the success of the experiment. Since several years the ROOT [5] team at CERN is developing a software framework, the Parallel ROOt Facility [6] (PROOF), which addresses the question of synchronous fast data processing on large computing farms. This framework is widely used in High Energy Physics (HEP) and in the ALICE experiment in particular.

The goal of PROOF is to allow for transparent interactive parallel analysis of large sets of files in ROOT format, a common container for data storage in HEP. It is conceived to provide transparency with respect to a local ROOT analysis session (same code can be run locally and in a PROOF system), scalability (no limitations on the number of machines that can be used in parallel) and adaptability (handling of changing of load, disk failure and network cut on the nodes). In this context, by *interactive* it is meant that the user is able to see the results right away, contrary to a Grid job where it is necessary to wait for the job to finish. *Parallel* means that the task is split into subtasks that will be executed on many computing nodes at the same time. Commonly we refer to analysis of *data* that are the result of events reconstruction (so called ESD, Event Summary Data or AOD, Analysis Object Data), but in the future PROOF can be extended to support also large scale simulation.

2 The PROOF system

PROOF is a system particularly suited to process events produced by high-energy physics experiments. In most analysis use cases events can be processed in an arbitrary order and partial results can be summed up after processing (trivial or event-based parallelism). It is an interactive system, therefore it can be used

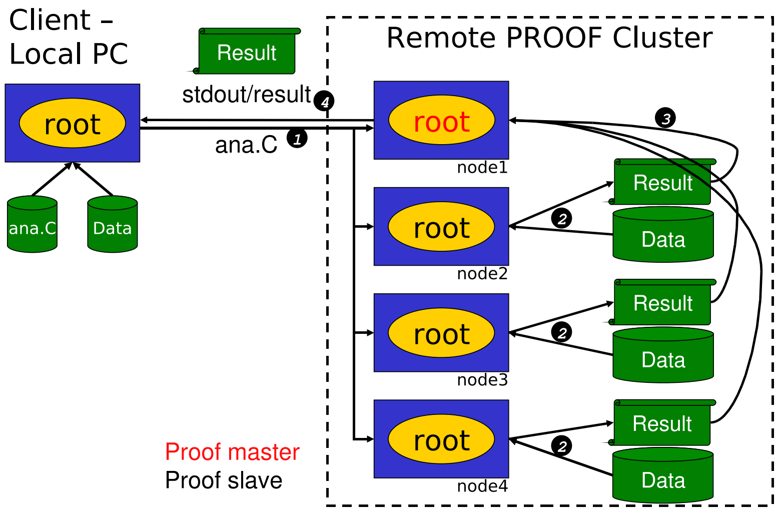


Fig. 1. Schema of the PROOF system. After the user connects to the cluster, a ROOT session is started on each node. The analysis code is sent to the head node (master) and then replicated on each worker. Each worker processes local data and produces a partial result that is sent back to the master node and merged together with the others. The final result is displayed on the user's screen.

from the ROOT prompt thus allowing for direct visualization of results. At the price of some overhead, result objects, e.g. histograms, can be monitored while they are being produced (so-called feedback histograms). Additional libraries, i.e. user code for processing, can be distributed with so-called PROOF packages.

The functioning schema of PROOF is shown in Fig. 1. A user running a ROOT session on her client connects to a PROOF master which in turn opens a ROOT session on each PROOF worker via Xrootd protocol [7]. Then the user submits a query, that consists of the analysis code and the list of files that she wants to be processed (step 1). The PROOF master splits the input dataset into data fragments and distributes them to the PROOF workers. The fragments are assigned in such a way that data local to a worker is processed first, then non-local data, if any remaining. After processing, the partial results are individually sent back to the PROOF master, merged together (step 3) and returned to the user (step 4). This workflow works automatically for mergeable results, that is the case for typical ROOT objects like histograms and trees. To this end, user objects must implement the merging functionality.

3 PROOF Concept on the Grid

PROOF is currently running for the ALICE experiment on a computing cluster called CAF [8] (CERN Analysis Facility) with 120 CPU cores. Naturally, a cluster of this size can only hold a fraction of the 3 PB data to be accumulated by the experiment. It is also not feasible, financial and support wise, to provide a computing capacity capable of handling the data volumes in one single computing centre. For these reasons the concept of PROOF necessitates to be extended to the Grid paradigm - the WLCG. Presently there are a number of research projects aiming at extending the PROOF functionality on the Grid. They are briefly introduced here below to clarify the conceptual and architectural differences as well as the goal each one wants to achieve.

– At INFN Torino, Italy, the Virtual Analysis Facility (VAF) project is currently running PROOF on a Grid Tier-2 (T2) cluster. In Grid terminology a T2 is a mid-size (few hundred CPU) computing centre for user analysis and MonteCarlo production.

  The cohabitation between batch and PROOF processes is achieved by running them on two separate virtual machines on Xen. When an interactive analysis must be prioritized with respect to batch jobs or vice versa, the operating system can directly suspend/restart or slow down/speed up an entire VM, transparently handling the processes memory footprint. Since running a VM requires administrator privileges, this kind of setup can be deployed at the level of a single computing centre, but not at the largest scale (the WLCG Grid) as in our case.

– The ATLAS experiment at CERN, in collaboration with the University of Wisconsin and BNL, USA, is developing a project to run Grid and PROOF services together on the same computing centre infrastructure using the

CONDOR [6] system to handle job priorities. CONDOR is a job scheduler mechanism allowing job submission in a local queue and provides command line API to fully control their behavior at scheduling and running time. As in Torino, the efforts are focused on how to run efficiently batch and interactive jobs on shared resources belonging to a single computing centre.

PROOF is an interactive analysis tool that, as such, may stay idle for a long period of time (for example nights and weekends). CONDOR allows for sharing the available resources between batch-like activities and PROOF, with the capability to get CPUs in a reasonably short time. In case interactive jobs are executed, running batch jobs are suspended freeing the used resources (CPU, I/O, memory) and resumed after the PROOF session is finished.

– At the GSI Heavy Ion Research Center in Darmstadt, Germany, the PROOF on Demand [9] package (PoD) is under development to perform PROOF-based distributed data analysis on the Grid and local batch systems. This successful project is already in production and has many common points with the result we want to achieve in ALICE: the ROOT/PROOF framework is used as a starting point, but the Grid access from ROOT is achieved using the gLite [10] implementation. PoD provides interfaces to submit Grid job scripts executed by the Local Resources Manager System (LRMS) on remote workers. These scripts, comparable to the pilot jobs in our proposal (section "System Architecture"), make environment recognition, upload of the necessary software packages and start the gLite-PROOF services.

  On the other hand there are two considerable differences in respect to our model that prevent its application. First of all the PROOF master is directly started on the user machine, i.e. each user connects to her own master. The choice to separate user environments has the advantage to obtain a robust architecture. The flip side is the assumption that PROOF workers may register themselves on the user machine: this is not certainly feasible on a Grid topology as the client machines are typically forbidden to accept incoming connections or neither allowed to ask for. The assumption a PROOF worker can directly register itself on the client machine leads to a 2-tiers architecture, whereas in our project we must add a gatekeeper on the site's front-end machine (optionally running also the local PROOF master), thus re-creating the 3-tiers architecture of a local PROOF cluster.

  The second difference resides in the dataset distribution. At GSI no assumptions are made on the data location, the access is up to the user code. On the contrary, the location of the file to be processed, that may be stored across several sites, is the keypoint of our computing model to run the code where data is, i.e. "bring the kB to the PB and not the PB to the kB".

In addition to the projects outlined above, PROOF is currently being used by a number of other HEP experiments. This is the case of the CMS collaboration in Oviedo, Spain, with the purpose to adapt the analysis framework to the PROOF model. Similarly, the LHCb collaboration at CERN has the aim to adapt in PROOF ad-hoc software for analysis, as well as the RHIC experiment at the Brookhaven National Laboratory, USA.

The extension of the PROOF concept to the Grid allows for interactive processing of large volumes of data distributed over hundreds of computing centres and accessible by many thousands of CPUs. The parallelization of the process has the obvious purpose of providing short response times. A number of problems (P) have been clearly identified, as well as proper solutions (S).

Cluster Connectivity

P The distributed PROOF clusters should be interconnected in a multi level hierarchy reflecting the PROOF cluster deployments.

S Each site will run its own PROOF master connected in turn to a general public PROOF superMaster acting as the starting point for the interactive user sessions. The distributed nature of the PROOF setup is hidden.

Tasks and Data colocation

P In a distributed computing environment the data sample to be analyzed will be located at many computing centres worldwide. The analysis tasks must be executed in the computing centre hosting the data, thus avoiding (heavy) data movement.

S This can be achieved by starting PROOF workers at the computing nodes of the centres through Grid jobs. The Grid middleware classes for asynchronous analysis allow for task splitting according to dataset location.

Protected Access

P The computing centres are protected by very strict access rules, implemented through complex firewalls, minimizing the methods that can be implemented for communication between tasks running in separate centres.

S In a classic PROOF local cluster, the head node initiates communications towards all the registered workers (master-to-worker). The worker nodes must be reachable from the PROOF master where the work is initiated. In a Grid topology a PROOF master is very likely running on the front-end machine of each centre, called VO-box, whereas the PROOF workers can communicate only through this VO-box. In this scenario the communication is reversed to a registration service running on the VO-box (worker-to-master).

Dynamic Scheduler

P The PROOF setup must be adapted to the dynamic Grid topology since potential computing workers and data servers may become available or drop out at any point in time, depending on the local availability of resources.

S The PROOF master must be able to connect to workers not only when the user starts a PROOF session. In the current PROOF implementation the number of workers is known a-priori. A dynamic worker allocation feature will require the development of a workload-based scheduler.

Interactivity

P The Grid is by implementation a batch system, where a job runs with a delay with respect of the submission time. With an interactive system, user tasks should start with the initiation of the analysis session.

S A number of Grid jobs will be always kept running in the local computing centres, ready to spawn PROOF sessions. The number of such jobs should be a function of the number of Grid users who start PROOF sessions concurrently and is adjusted automatically.

4 System Architecture

By implementing the above mentioned solutions we can achieve a novel way to run the PROOF setup on distributed resources accessing large data volumes, preserving at the same time the key benefits to run interactive and parallel data processing. Fig. 2 displays the system architecture in a Grid environment.

A ProxyServer service is running on a public port on each VO-box (step 1). This module is installed together with the Grid services deployed on the front end machine of the given site. A number of pilot Grid jobs (step 2) based, among

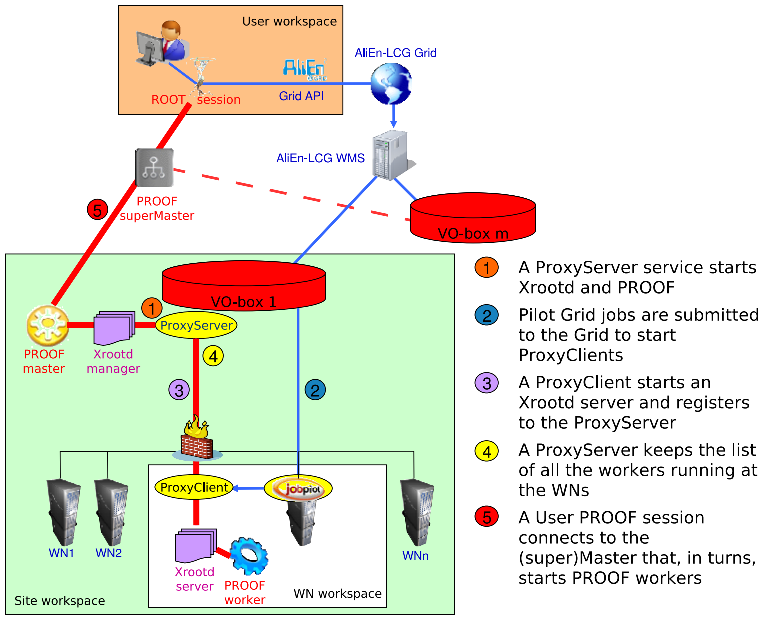

User workspace
AliEn-LCG Grid
ROOT session
Grid API
AliEn-LCG WMS
PROOF superMaster
5
VO-box m
VO-box 1
1
ProxyServer
PROOF master
Xrootd manager
4
3
2
ProxyClient
jobpilot
WN1
WN2
WNn
Xrootd server
PROOF worker
WN workspace
Site workspace
1 A ProxyServer service starts Xrootd and PROOF
2 Pilot Grid jobs are submitted to the Grid to start ProxyClients
3 A ProxyClient starts an Xrootd server and registers to the ProxyServer
4 A ProxyServer keeps the list of all the workers running at the WNs
5 A User PROOF session connects to the (super)Master that, in turns, starts PROOF workers

Fig. 2. Schema of the PROOF system on the Grid. New components need to be plugged into the system to allow dynamic usage of loosely-coupled distributed resources. The PROOF architecture master-to-slave is inverted to work through fairly strict firewalls preventing direct connections to computing machines at the remote Grid sites.

the others, on the cardinality and location of the user input dataset, is submitted to the Grid from the user interface with the purpose to start a certain number of ProxyClient, the counterpart of the ProxyServer. Pilot jobs are created by the task splitting capabilities of the Grid middleware and sent to the computing sites close to the Storage Element(s) (SEs) hosting the dataset. The number of pilot jobs is a function of the number of files the user wants to process and their distribution among the SEs.

ProxyClient services already running can be re-used for further user sessions. In such way, the latency of the Grid is hidden because the proxies are kept running once started, ready to serve new tasks. A ProxyClient starts an Xrootd server from the proper ROOT package installed at the site. Users might ask for different versions of a given ROOT/Xrootd package: if it is not present on the Grid Worker Node (WN), it will be automatically downloaded using the Grid Package Manager service. Whenever a ProxyClient process is started on a WN, it registers itself on the ProxyServer (step 3) and establishes an outgoing connection towards the Grid VO-box. The ProxyServer acts as a gatekeeper and keeps the list of all ProxyClients running on the WNs (step 4).

When a user starts a ROOT session (step 5) and connects to the distributed PROOF cluster, the proper workers are selected among the available ones. The local PROOF master at the VO-box accepts connections from the public PROOF superMaster (running on a public machine and coordinating the activity of the local masters) and, through the proxies, connects the PROOF workers at the protected sites (red thick line). The PROOF superMaster distributes the load among the PROOF clusters started at the Grid sites and shields the user from the underlying complexity.

A prototype of the distributed PROOF framework is currently under development and test on two Grid sites at CERN and NIHAM (Romania). These sites store recent ESD data generated by the production cycles of the ALICE Physics Data Challenge 2008 and 2009 (PDC08/09). As a proof of concept, simple analysis tasks processing ESD and reading Monte Carlo tracks have been successfully executed (Fig. 3). The current challenge consists in connecting the individual Grid sites throughout the same session whenever an input dataset is spread over different Grid SEs. Meanwhile, the robustness of the ProxyServer must be improved to support a higher number of connections (tested up to 20 TCP sockets per site). Performance tests are in progress to precisely determine the increment of load sustained by the VO-box with the introduction of the PROOF master. It is well known that this service produces negligible traffic as long as data is processed by the workers (it must only coordinate the job among them) but has peaks of CPU usage and memory consumption during the final merging. The understanding of the PROOF master performance and scalability will come with the integration of the ProxyServer in the MonALISA monitoring system [11] in use in ALICE. The ProxyServer will plug-in into the MonALISA distributed agents already deployed at each Grid site with the advantage to be completely monitored and remotely managed. MonALISA provides the capacity

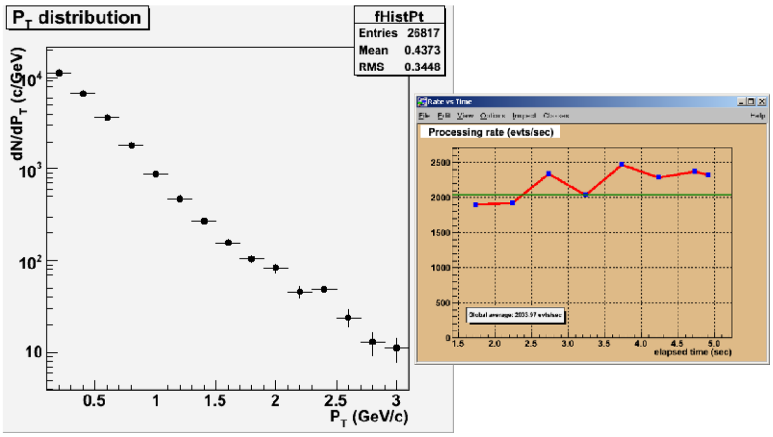


Fig. 3. Analysis task running in distributed PROOF. The plot displays the Pt spectrum out of 26k tracks with the corresponding processing rate (events/sec).

to send monitoring information to the central ALICE Web repository [12] for history view or an interactive GUI client for detailed short-time views.

5 Summary

This contribution presents the approach of the ALICE experiment to enable fast data analysis on the Grid middleware in usage, the ALICE Environment (AliEn). The PROOF framework is primarily meant as an interactive alternative to batch systems for central analysis facilities and departmental workgroups (Tier-2s) in particle physics experiments. However, thanks to a multi-tier architecture allowing multiple levels of masters, it can be adapted to a wide range of virtual clusters distributed over geographically separated domains and heterogeneous machines that form the Grid. The dynamic allocation of a Grid Analysis Facility running PROOF allows to quickly prototype user code that needs many iterations on input datasets, with the advantage given by the availability of the entire data production of the experiment instead of only a small subset locally staged. To this end, new components must be plugged into the system, i.e. a tunnel between the Grid WNs and corresponding VO-box to work through firewall protections, an hierarchy of PROOF masters coordinating the work among the Grid sites, a PROOF workers distribution based on the AliEn capabilities to split jobs according to the data location and a dynamic allocation of PROOF workers.

Acknowledgments

The author would like to thank Jan Fiete Grosse-Oetringhaus for the good teamwork during the CAF setup and maintenance in the ALICE Offline group at CERN, Latchezar Betev and Federico Carminati for the support in this project and review of this document.

References

1. ALICE Technical Proposal for a Large Ion Collider Experiment at the CERN LHC. CERN/LHCC/95-71, December 15 (1995)
2. Saiz, P., Aphecetche, L., Buncic, P., Piskac, R., Revsbech, J.E., Sego, V.: AliEn - ALICE environment on the GRID. In: Nuclear Instruments and Methods 2003, pp. 437–440 (2003), http://alien.cern.ch
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. The Worldwide LHC Computing Grid Project http://lcg.web.cern.ch/LCG
5. Brun, R., Rademakers, F.: ROOT - An Object Oriented Data Analysis Framework. Nucl. Instr. And Meth. A 502, 339–346 (2003)
6. Ballintijn, M., Roland, G., Brun, R., Rademakers, F.: The PROOF distributed parallel analysis framework based on ROOT. In: Proc. Conf. for Computing in High-Energy and Nuclear Physics (CHEP), La Jolla, California, 24-28 March (2003)
7. The Scalla Software Suite: xrootd/cmsd, http://xrootd.slac.stanford.edu
8. Grosse-Oetringhaus, J.F.: The CERN analysis facility: A PROOF cluster for day-one physics analysis. J. Phys. Conf. Ser. 119, 072017 (2008)
9. Manafov, A.: PROOF on Demand - An implementation of the PROOF distributed data analysis on different batch systems and/or on the Grid, https://subversion.gsi.de/trac/dgrid
10. http://glite.web.cern.ch/glite/
11. Legrand, I.C., Newman, H.B., Voicu, R., Cirstoiu, C., Grigoras, C., Toarta, M., Dobre, C.: MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications. In: CHEP 2004, Interlaken, Switzerland (September 2004), http://monalisa.caltech.edu/monalisa.htm
12. Meoni, M.: Monitoring of a distributed computing systems: the Grid AliEn@CERN, University of Florence, Italy, December 19 (2005), http://monalisa.caltech.edu/docs/marco_thesis.pdf, MonALISA Repository for ALICE, http://pcalimonitor.cern.ch

A Comparison of Performance of Sequential Learning Algorithms on the Task of Named Entity Recognition for Indian Languages

Awaghad Ashish Krishnarao, Himanshu Gahlot, Amit Srinet, and D.S. Kushwaha

Motilal Nehru National Institute of Technology,
Allahabad – 211004,
India
{awaghad.ashish,himanshu.gahlot86,amit.vers}@gmail.com,
dsk@mnnit.ac.in

Abstract. We have taken up the issue of named entity recognition of Indian languages by presenting a comparative study of two sequential learning algorithms viz. Conditional Random Fields (CRF) and Support Vector Machine (SVM). Though we only have results for Hindi, the features used are language independent, and hence the same procedure could be applied to tag the named entities in other Indian languages like Telgu, Bengali, Marathi etc. that have same number of vowels and consonants. We have used CRF++ for implementing CRF algorithm and Yamcha for implementing SVM algorithm. The results show a superiority of CRF over SVM and are just a little lower than the highest results achieved for this task. This can be attributed to the non-usage of any pre-processing and post-processing steps. The system makes use of the contextual information of words along with various language independent features to label the Named Entities (NEs).

Keywords: Support Vector Machines, Conditional Random Fields, Maxent, NER, Hindi, Named Entities.

1 Introduction

Named Entity Recognition for European Languages has already achieved a significant accuracy (Sang, 2002 [6]; Sang and De Meulder, 2003 [5]), which is especially high for English, and even for East Asian languages (Sassano and Utsuro, 2000 [16]). But the same task for Indian languages like Devnagiri (Hindi) is lagging far behind due to its various intricacies like, missing capitalizatoin information, which is the single most important information for identifying NEs. Indian languages also lack a formal and large list of gazetteers which makes pre-processing inefficient. Indian languages also have the problem of disambiguation of common nouns from proper nouns. A fairly large no. of frequently used words (common nouns) in Indian languages can also be used as named entities. For example, words like *Vivek, Kamal, Pankaj, Deepak* etc. which are proper nouns can also confer some meaning and can be used as common nouns. While in english almost all proper nouns are meaningless viz. *Harry, Kate, Leonardo, Jessica* etc. Hence the disambiguation in usage of a word as common noun or proper noun using its contextual features in Indian languages is more difficult and important than that for European languages.

Due to the above mentioned problems NER task for Indian languages is more difficult. Hence, even though at least four of them figure in the top ten spoken languages of the world these languages are less studied by researchers. Our comparative study of NER for these languages using two sequential labelling algorithms will provide an insight to researchers to work on the best method and shape their research accordingly. As opposed to the four tag tagset used in CoNLL 2003 shared task we have used a tweleve tag tagset which was used in NERSSEAL-08 shared task. The system first identifies the named entities and then predicts their correct tags using the model generated through training.

2 Previous Work

The latest research work on named entity recognition for Indian languages was reported in the NERSSEAL-08 shared task against a baseline that uses maximum entropy based name finder tuned for English but trained on data from five South Asian languages viz. Hindi, Telugu, Bengali, Oriya and Urdu. The highest F-measure of 65.13% was reported by Saha et al. while the second highest was 50.06%, reported by Karthik et al. Our tagset and test data is the same as was used in this workshop.

Mainly two approaches are applied for NER :

- Linguistic approach
- Machine Learning approach

Linguistic approach works on handcrafted rules which are written by skilled linguists. Previous rule based NER systems, containing mainly lexicalized grammar, gazetteer lists, and list of trigger words, are capable of providing upto 92% F-measure accuracy for English (Grishman, 1995 [8]; McDonald, 1996 [17]). These systems have a disadvantage that they require huge experience and grammatical knowledge of the particular language or domain and are not transferable to other languages or domains. However, given the closer nature of many Indian languages, the cost of adaptation of a resource from one language to another could be quite less (Singh and Surana, 2007 [2]).

The machine learning techniques tried for NER include the following:

- Hidden Markov Models or HMM (Zhou and Su, 2001 [7])
- Decision Trees (Isozaki, 2001 [9])
- Maximum Entropy (Borthwick et al., 1998 [1])
- Support Vector Machines or SVM (Takeuchi and Collier, 2002 [11])
- Conditional Random Fields or CRF (Settles, 2004 [12], Li and Mccallum, 2004 [15])

These techniques make use of a large amount of NE annotated training data to acquire high level language knowledge.

Srihari et al. (2000) [19] used a combination of different ML approaches.They combined MaxEnt, HiddenMarkov Model (HMM) and handcrafted rules to build an NER system. Such hybrid systems have been generally more effective at the task of NER.

Previous work on NER for Indian and some other South and South East Asian Languages has been done by McCallum and Li, 2003 [3] and Cucerzan and Yarowsky, 1999 [18] but this work was just an extension of the work done on European Languages. Cucerzan and Yarowsky [18] in their language independent NER work used morphological and contextual evidences. They ran their experiment with 5 languages - Romanian, English, Greek, Turkish and Hindi. Among these the accuracy for Hindi was the worst. For Hindi the system achieved 41.70% F- measure with a very low recall of 27.84% and about 85% precision. The F-measure of the system developed by Wei Li and Andrew McCallum (2004) [15] was a lot better, giving an F measure of 71.50% against a training data of 340000 words. The maximum accuracy for NER in Hindi is reported by Kumar and Bhattacharyya, (2006) [14]. They achieved an F measure of 79.7% using a Maximum Entropy Markov Model.

3 Named Entity Tagset

We have used a twelve tag tagset for our classification task. This tagset gives a finer classification which in turn is helpful for better machine translation task. These tags are briefly explained in Table 1.

Table 1. The twelve tag tagset used in our task of named entity recognition

Tag	Name	Description
NEP	Person	Deepak, Gandhi
NED	Designation	Secretary, Director
NEO	Organization	Municipal Corporation
NEA	Abbreviation	C.E.O., A.D.A.
NEB	Brand	Honda, Reebok (ambiguous)
NETP	Title-Person	Mr., Mrs., Dr.
NETO	Title-Object	Hamlet, Othello
NEL	Location	Mumbai, Bangalore, Delhi
NETI	Time	18th September, 1984 (ambiguous)
NEN	Number	7.94, 8.56, 420
NEM	Measure	4 grams, Rs 4,000
NETE	Terms	Botany, Maximum Entropy

4 Data Description and Features Used

The data for our task was in *Shakti Standard Format (SSF)* which is shown below :

```
0      ((        SSF
1.     ((        NP  <ne=NEP>
1.1    Mahatma
1.2    Gandhi
       ))
2.     ke
```

3. kehne
4. par
))

We have converted this data to BIO format (B=Begin of chunk, I=Inside chunk, O=Outside the chunk) as follows:

Mahatma	B-NEP
Gandhi	I-NEP
Ke	O-NOT
kehne	O-NOT
par	O-NOT

The input format for both CRF++ and Yamcha SVM is same i.e. the word to be classified is followed by its features and then the class label. The initial training data for our classification task consisted of 503179 instances (words). The testing data had 40543 instances. This data had too many sentences which did not contain even a single named entity hence we refined the data by eliminating these sentences. Hence our refined data had all those sentences which had one or more named entities. The refined data had 301620 instances. We have used the following features :

- Word Window – A window size of 3 has been used to take into account the words in context of the current word.
- Prefix information – Prefix information upto length 3 is used.
- Suffix information – Suffix information upto length 3 is used.
- Length of the word – Word length information is used.
- Sentence start information – Whether the current word is the start of the sentence or not.
- Two consecutive digits – Occurrence of two consecutive numbers is quite frequent and hence it served as a good feature.
- Four consecutive digits – It mainly indicated the occurrence of 'years'. Served as a good feature.
- Word class and Brief class - Words are also assigned a generalized 'word class (WC)', which replaces all letters with 'a', digits with '0', punctuation marks with 'p', and other characters with '-'. There is a similar 'brief class (BWC) (Settles 2004 [12])' which collapses consecutive characters into one. Thus the words "D.D.T." and "AB-1946" would both be given the features WC=apapap, BWC=apapap.

5 Learning Algorithms Applied

We used two learning algorithms for our classification task : Conditional Random Fields (CRFs) (Lafferty et al., 2001 [10]) and Support Vector Machines (SVMs) (Cortes and Vapnik, 1995 [4]). CRFs are implemented using CRF++ while SVMs are implemented using Yamcha. These algorithms are discussed below.

5.1 Conditional Random Fields

Conditional Random Fields (CRFs) are undirected graphical models used to calculate the conditional probability of values on designated output nodes given values assigned to other designated input nodes.

In the special case in which the output nodes of the graphical model are linked by edges in a linear chain, CRFs make a first-order Markov independence assumption, and thus can be understood as conditionally trained finite state machines (FSMs). Let $o = (o_1,o_2,o_3 ,o_4 ,... o_T)$ be some observed input data sequence, such as a sequence of words in text in a document,(the values on n input nodes of the graphical model). Let S be a set of FSM states, each of which is associated with a label, l Є £.

Let $s = (s_1,s_2 ,s_3 ,s_4 ,... s_T)$ be some sequence of states, (the values on T output nodes). By the Hammersley Clifford theorem, CRFs define the conditional probability of a state sequence given an input sequence to be:

$$P(s|o) = 1/Z_0 *exp(\sum_{t=1}^{T} \sum_{k} \lambda_k f_k(s_{t-1}, s_t, o, t)) .$$

where *Zo* is a normalization factor over all state sequences is an arbitrary feature function over its arguments, and *λk* is a learned weight for each feature function. A feature function may, for example, be defined to have value 0 or 1. Higher *λ* weights make their corresponding FSM transitions more likely. CRFs define the conditional probability of a label sequence based on the total probability over the state sequences,

$$P(l|o) = \sum_{s:l(s)=l} P(s|o) .$$

where *l(s)* is the sequence of labels corresponding to the labels of the states in sequence s.

Note that the normalization factor, Zo, (also known in statistical physics as the partition function) is the sum of the scores of all possible states.

$$Z_0 = \sum_{s \varepsilon s^T} exp(\sum_{t=1}^{T} \sum_{k} \lambda_k f_k(s_{t-1}, s_t, o, t)) .$$

And that the number of state sequences is exponential in the input sequence length, T. In arbitrarily-structured CRFs, calculating the partition function in closed form is intractable, and approximation methods such as Gibbs sampling or loopy belief propagation must be used. In linear-chain structured CRFs (in use here for sequence modeling), the partition function can be calculated efficiently by dynamic programming.

5.2 Support Vector Machines

Support Vector Machines (SVMs) are supervised machine learning algorithm for binary classification on a feature vector space x ε R^L.

$$w . x + b = 0, \qquad w \varepsilon R^L, \ b \varepsilon R . \tag{1}$$

Suppose the hyperplane (1) separates the training data, $\{(xi, yi) \mid xi \varepsilon R^L, \ y_i \varepsilon \{\pm 1\}, 1 \leq i \leq l \}$, into two classes such that

$$y_i.(w.x_i + b) \geq 1 . \tag{2}$$

While several of such separating hyperplanes exist, SVMs find the optimal hyperplane that maximizes the margin (the distance between the hyperplane and the nearest points). Such a hyperplane is known to have the minimum expected test error and can be solved by quadratic programming. Given a test example x, its label y is decided by the sign of discriminant function $f(x)$ as follows:

$$f(x) = w . x + b, \tag{3}$$
$$y = \text{sgn}(f(x)), \tag{4}$$

For linearly non-separable cases, feature vectors are mapped into a higher dimensional space by a nonlinear function $\Phi(x)$ and linearly separated there. In SVMs' formula, since all data points appear as a form of inner product, we only need the inner product of two points in the higher dimensional space. Those values are calculated in R^L without mapping to the higher dimensional space by the following function $K(x_i, x_j)$ called a kernel function,

$$\Phi(x_i) . \Phi(x_j) = K(x_i, x_j) , \tag{5}$$

Since SVMs are binary classifiers, we must extend them to multi-class classifiers to predict $k > 2$ POS tags.

Among several methods of multi-class classification, we employ the one-versus-rest approach. In training, k classifiers $f_i(x)$ $(1 \leq i \leq k)$ are created to classify the class i from all other classes,

$$\begin{array}{ll} f_i(x) \geq +1 & \text{x belongs to the class } i, \\ f_i(x) \leq -1 & \text{otherwise.} \end{array}$$

Given a test example x, its class c is determined by the classifier that gives the largest discriminating function value,

$$c = \text{argmax } f_i(x) \tag{6}$$

6 Results

Table 2 shows the resulting classification of individual tags using both CRFs and SVMs on full data, on full data using a word window and on refined data. It is clearly evident from the results that a refined training data i.e. a data consisting of only those sentences which have at least one named entity outperforms the accruacy

Table 2. Comparison of CRFs and SVMs with full data and refined data. Refined data shows a better accuracy than full data while overall CRFs outperform SVMs.

Algo	**Tag**	**NEP t=245**	**NED t=72**	**NEO t=103**	**NEA t=8**	**NETO t=325**	**NEL t=235**	**NETI t=92**	**NEN t=514**	**NEM t=31**	**NETE t=1080**	**NEB t=0; NETP t=5**
	Data	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*
CRFs	**Full Data**	90	20	26	2	12	110	53	368	14	90	0
	Word Window Four	94	21	23	3	18	109	53	364	15	90	0
	Ref. Data	98	19	27	3	36	129	59	421	17	127	0
SVMs	**Full Data**	33	18	10	1	8	62	34	326	13	51	0
	Word Window Four	38	17	10	1	9	63	36	333	13	67	0
	Ref. Data	48	17	12	1	9	70	37	367	13	76	0

t = total count ; c = correct count; Full Data = 503179 instances ;
Ref. Data = Refined Data = 301620 instances; Experiment with word window four is performed on full data.

of the data having a large number of sentences without a single named entity. It is also clearly evident that CRFs outperform SVMs when it comes to Named Entity Resolution.

6.1 Calculation of F-Measure, Precision and Recall

We present our result on three measures of performance calculated for three cases : maximal named entities, nested named entities and lexical matches. Thus, there are nine measures in total :

- Maximal Precision: $P_m = C_m / R_m$
- Maximal Recall: $R_m = C_m / T_m$
- Maximal F-Measure: $F_m = (2 * P_m * R_m) / (P_m + R_m)$
- Nested Precision: $P_n = C_n / R_n$
- Nested Recall: $R_m = C_n / T_n$
- Nested F-Measure: $F_n = (2 * P_n * R_n) / (P_n + R_n)$
- Lexical Precision: $P_l = C_l / R_l$
- Lexical Recall: $R_l = C_l / T_l$
- Lexical F-Measure: $F_l = (2 * P_l * R_l) / (P_l + R_l)$

where *C* is the number of correctly retrieved (identified) named entities, *R* is the total number of named entities retrieved by the system being evaluated (correct plus incorrect) and *T* is the total number of named entities in the reference data.

Table 3 lists these measures for various experiments performed.

Table 3. The nine performance measures for each experiment (P=Precision; R=Recall)

Algo	**Type**	**Maximal**			**Nested**			**Lexical**		
	Data	**P**	**R**	**Fβ**	**P**	**R**	**Fβ**	**P**	**R**	**Fβ**
CRFs	**Full Data**	0.55	0.27	0.37	0.58	0.27	0.37	0.79	0.29	0.42
	Word Window four	0.56	0.27	0.37	0.60	0.27	0.37	0.81	0.29	0.43
	Ref. Data	0.52	0.32	0.40	0.56	0.32	0.40	0.76	0.34	**0.47**
SVMs	**Full Data**	0.61	0.21	0.31	0.66	0.20	0.31	0.86	0.20	0.33
	Word Window four	0.56	0.23	0.35	0.57	0.22	0.32	0.83	0.26	0.35
	Ref. Data	0.59	0.25	0.35	0.63	0.24	0.35	0.82	0.24	0.37

7 Error Analysis

As the results show the f-value measure is highest for CRFs using a refined data while the SVMs give the lowest f-value measure. These results are obtained against a base line of 25.68%. We can easily infer that the sparseness of the named entities plays a major role in deciding the final classification. Also, as a finite state machine derived from HMMs, CRFs can naturally consider state-to-state dependences and feature-to-state dependences. On the other hand, SVMs do not consider such dependencies. SVMs separate the data into categories via a kernel function. They implement this by mapping the data points onto an optimal linear separating hyperplane. Finally, SVMs do not behave well for large number of feature values. For large number of feature values, it would be more difficult to find discriminative lines to categorize the labels.

In recognition task, NETO and NETE are giving worst results since they are difficult to discriminate even manually. The low results of the whole process are due to the use of ambiguous and a vast variety of named entity tags.

8 Conclusion

Since the Indian sub-continent has a large variety of regional languages which inherently have the same number of vowels and consonants and also the semantic of construction of words, the results give future researchers a direction regarding the suitability of the algorithms and also to focus their research in the areas of shortcomings viz. on the ambigous tags like NETO and NETE as also the usage of gazetteers and other pre-processing and post-processing steps to refine the results. Nevertheless, we feel that our results are far better since we have achieved higher accuracy without any pre-processing or post-processing steps and it is expected to increase the efficiency of the system with these aids.

References

1. Borthwick, A., Sterling, J., Agichtein, E., Grishman, R.: Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In: Proceedings of the Sixth Workshop on Very Large Corpora, pp. 152–160 (1998)
2. Singh, A.K., Surana, H.: Can Corpus Based Measures be Used for Comparative Study of Languages? In: Proceedings of Ninth Meeting of the ACL Special Interest Group in Computational Morphology and Phonology, ACL (2007)
3. McCallum, A., Li, W.: Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In: Seventh Conference on Natural Language Learning (CoNLL) (2003)
4. Cortes, C., Vapnik, V.: Support-vector network. Machine Learning 20, 273–297 (1995)
5. Sang, E.F.T.K., De Meulder, F.: Language Independent Named Entity Recognition. In: Introduction to the CoNLL 2003 Shared Task. Development, vol. 922, p. 1341 (2003)
6. Erik, F., Kim Sang, T.: Introduction to the conll 2002 shared task: Language-independentnamed entity recognition. In: Proceedings of CoNLL 2002, Taipei, Taiwan, pp. 155–158 (2002)
7. Zhou, G.D., Su, J.: Named entity recognition using an HMM-based chunk tagger. In: Proceedings of the 40th AnnualMeeting on Association for Computational Linguistics, pp. 473–480 (2001)
8. Ralph, G.: The New York University System MUC-6 orWhere's the syntax? In: Proceedings of the Sixth Message Understanding Conference (1995)
9. Isozaki, H.: Japanese named entity recognition based on a simple rule generator and decision tree learning. In: Meeting of the Association for Computational Linguistics, India, pp. 306–313 (January 2001)
10. John, L., Andrew, M., Fernando, P.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001) (2001)
11. Takeuchi, K., Collier, N.: Use of support vector machines in extended named entity recognition. In: Proceedings of the sixth Conference on Natural Language Learning (CoNLL 2002) (2002)
12. Settles, B.: Biomedical Named Entity Recognition Using Conditional Random Fields and Rich Feature Sets. log, 1:1
13. Karthik, G., Harshit, S., Ashwini, V., Praneeth, S., Misra, S.D.: Aggregating machine learning and rule based heuristics for named entity recognition. In: Proceedings of the IJCNLP 2008 Workshop on NER for South and South East Asian Languages, Hyderabad, India, pp. 25–32 (January 2008)
14. Kumar, N., Pushpak, B.: Named Entity Recognition in Hindi using MEMM. Technical Report, IIT Bombay, India (2006)
15. Wei, L., Andrew, M.: Rapid Development of Hindi Named Entity Recognition using Conditional Random Fields and Feature Induction. ACM Transactions on Computational Logic (2004)
16. Sassano, M., Utsuro, T.: Named entity chunking techniques in supervised learning for japanese named entity recognition. In: Proceedings of the 18th conference on Computational linguistics, Morristown, NJ, USA, pp. 705–711. Association for Computational Linguistics (2000)

17. McDonald, D.: Internal and external evidence in the identification and semantic categorization of proper names. In: Boguraev, B., Pustejovsky, J. (eds.) Corpus Processing for Lexical Acquisition, pp. 21–39 (1996)
18. Cucerzan, S., Yarowsky, D.: Language independent named entity recognition combining morphological and contextual evidence. In: Proceedings of the Joint SIGDAT Conference on EMNLP and VLC 1999, pp. 90–99 (1999)
19. Srihari, R., Niu, C., Li, W.: A Hybrid Approach for Named Entity and Sub-Type Tagging. In: Proceedings of the sixth conference on Applied natural language processing (2000)

Managing Multi-concern Application Complexity in AspectSBASCO

Manuel Díaz, Sergio Romero, Bartolomé Rubio, Enrique Soler, and José M. Troya

Department of Languages and Computer Science, University of Málaga, 29071 Spain
{mdr,sromero,tolo,esc,troya}@lcc.uma.es

Abstract. AspectSBASCO is a new programming environment that integrates modern technologies (i.e. software components, parallel skeletons and aspects) to support the development of parallel and distributed scientific applications. This multi-paradigm approach provides high-level parallelism, reusability, interoperability and a clearer separation of concerns. This paper is focussed on a case study in which the programming model of AspectSBASCO is applied for the efficient solution of a relatively complex reaction-diffusion problem. In the application, the system of non-linear PDEs is solved in parallel using skeleton abstractions for domain decomposition methods. In addition, other concerns including distributed simulation persistence, mesh adaptation procedures, dynamic processor re-mapping and state variables communication are implemented in a modular way using (un)pluggable aspects. This style of application development leads to a better system decomposition, which is the key to improving software evolution, maintenance, productivity and reliability.

1 Introduction

Software engineering advances in sequential computing are often difficult to spread on the high-performance computing (HPC) scene, where parallel programming models have to deal with an additional dimension of complexity (i.e. concurrency), and where portability is restricted by the types of underlying hardware architectures. Examples of relatively recent technologies which are proven to achieve high-quality decomposition of sequential and distributed applications are those based on software components and aspects.

Component-oriented programming (COP) [8] is the paradigm that proposes the construction of applications plugging stand-alone and reusable pieces of software, so called components. However, the most extended component models and implementations (e.g. Microsoft COM+, Sun EJB, OMG CCM) lack the abstraction needed for scientific applications. For example, a component in these models cannot encapsulate (at least, in a natural way) a parallel application which must be executed on a group of processors and interact efficiently with other processors. For this reason, specific solutions for HPC have emerged in the last few years. Some examples are CCA [1] and ASSIST [14].

When designing and building complex systems (even when components are used) it is difficult to produce designs that modularize all the system requirements. Typically, there are some characteristics that do not fit well into any component structure chosen. This is particulary true for concerns such as logging, debugging, communication, synchronization, security, and so on. Design alternatives often lead to code where the same concern spreads over (i.e. cross-cuts) many system modules. Aspect-oriented programming (AOP) [9] enables developers to capture the cross-cutting structure so that concerns can be programmed in a centralized way. Although AOP is very extended for sequential application development, the paradigm is hardly ever applied to HPC. Some interesting work is that of [7] and [12], both based on using the popular aspect-oriented language AspectJ for the modularization of high-performance concerns.

In a different context, structured parallel programming [10] proposes the use of skeletons, which are reusable parallelism exploitation patterns. The idea behind skeletal programming is to provide the programmer with a collection of predefined parallelism constructors (the skeletons) which can be combined to declaratively express the parallel structure of the application. Using this paradigm the developer is free from implementing low-level operations (e.g. task creation, data communication) and thereby she/he can focus on the numerical algorithms. Muskel [3] and eSkel [2] are representative examples of skeleton-based systems.

AspectSBASCO [6] is a new programming environment for HPC applications which provides a multi-paradigm approach that combines the above-mentioned technologies (i.e. software components, parallel skeletons and aspects). Components and aspects have been previously unified successfully in work dealing with non-parallel models [13]. Components and skeletons were used together in some parallel approaches [14]. However, and to the best of our knowledge, our proposal is the first attempt to combine the three technologies in the context of HPC.

This paper presents a case study on applying the AspectSBASCO programming model for the efficient solution of a complex reaction-diffusion problem, which is solved using skeleton abstractions for parallel domain decomposition methods [11]. Additional application concerns are implemented using aspects (which can be plugged, or unplugged, if necessary). Some examples are distributed simulation persistence, mesh refinement and dynamic processor re-mapping, which aim for improvements in different directions (e.g. accuracy, performance). Using AspectSBASCO, these kinds of "extra-functional" concerns of vital importance in scientific applications can be separate from the numerical code and become easier to develop and evolve as they are well-modularized.

A similar problem, though considering a simpler set of application requirements, was previously studied using SBASCO [4][5], the predecessor of AspectSBASCO which only combined coarse-grain scientific components and skeletons.

2 Overview of AspectSBASCO

There are two types of components in AspectSBASCO: Scientific Components (SCs) which implement the tasks that solve the numerical problem, and Aspect

Components (ACs) for encapsulating the cross-cutting functionality in applications. In addition, a family of three parallel skeletons is defined: *multiblock* is used for the solution of domain decomposition and multi-block problems; *farm* improves the throughput of a task as different data sets can be processed in parallel; *pipe* enables a sequence of tasks which can be executed concurrently to be pipelined. A detailed explanation of these skeletons can be found in [4].

Scientific components can implement sequential or parallel tasks. A parallel SC can use skeleton declarations to establish its internal parallel structure in a high-level way. Otherwise, the developer is free to implement "ad-hoc" parallelism in the component (e.g. unstructured, data-parallelism). Skeletons are not only used for internal component structure but also for application parallelism. For instance, an application can consist of a group of SCs which interact following the multiblock paradigm. Skeletons represent an elegant and declarative style of integrating different parallelism types (e.g. task and data parallelism).

The interaction between SCs follows a data-flow style and is based on two communication primitives called `get_data()` and `put_data()`. The *configuration interface* of a SC influences this type of communication. This interface describes the input/output arguments, their data-distribution, the processor layout and, (if possible) the internal structure in terms of skeleton declarations. Exposing this kind of information at the component interface level enables the system to implement efficient data communication and task interaction.

Aspect components implement the aspect code which will be executed at different points (join points) of the application control flow. Interactions between SCs and ACs are based on method calls (instead of data-flow). An AC can provide one or several interfaces of operations describing aspect functionality. A new type of elements called Aspect Connectors (ACNs) encapsulate the SC-AC interaction information. Specifically, an ACN indicates the methods on the corresponding ACs to be invoked at specific pointcuts (namely subsets of join points) that refer to the participant SCs. Expressing the interaction information in a separate layer of ACNs improves component development and reuse.

ACN declarations exploit the AspectSBASCO join point model which defines valid points for the potential execution of aspect code. The union of two different sets of points is considered. First, some internal actions identified as generic for our skeleton-based applications (e.g. calculation of new time step, communication based on `get_data()` and `put_data()`), and second, any method call carried out on *external components*, which are components that represent functionality reused by several SCs (for instance, a wrapper to a legacy code library).

The implementation of an AC may require access to the internal state of the SCs. For this reason, the latter can define additional interfaces that expose SC internal variables and properties. These interfaces are so-called *aspect interfaces* and are described using a subset of the OMG IDL language. Aspect interfaces are designed to enable the plugging and execution of aspects by means of the traditional *provide-use* pattern followed in component models such as CCM.

More detailed descriptions of the mechanisms that enable the definition and execution of aspects in our programming model can be found in [6].

3 Case Study: Reaction-Diffusion Equations

This section describes the numerical problem, the application design in AspectS-BASCO and some issues regarding implementation.

3.1 Numerical Problem Definition

The reaction-diffusion problem considered is characterized by a system of two time-dependent PDEs which are coupled by a non-linear source term. Basic combustion and heat transfer phenomena can be modeled as follows.

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + S(U), \qquad U = (u, v)^T, \qquad S = (-uv, uv - \lambda v)^T \tag{1}$$

The variables u and v represent the concentration of a reactant and the temperature, respectively, t is time, x and y denote Cartesian coordinates, λ is a constant (in this paper, $\lambda = 0.5$), and the superscript T denotes transpose.

Equation (1) was discretized in time and space using finite differences and implicit linearized θ-method, where the non-linear term $S_{i,j}^{n+1}$ was approximated using a Taylor polynomial to obtain a system of linear algebraic equations.

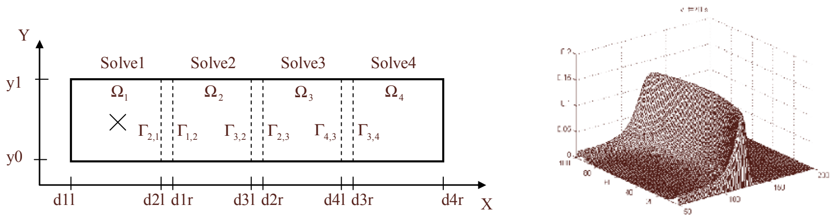


Fig. 1. Decomposition of the problem using four overlapping domains (left) and graphical representation of variable v, temperature, at $t = 70$ (right)

The problem is studied on a two-dimensional geometry similar to the one shown in Fig. 1 (left). The point marked with a cross denotes the ignition point, which is the temperature peak needed to start the reaction. The same figure depicts a snapshot of the solution, which corresponds to a traveling reaction wave front that propagates from the ignition point to the rest of the domain.

The numerical solution is calculated by means of a overlapping domain decomposition (i.e. Schwarz) method which admits parallel processing. The algorithm used proceeds as follows: the geometry is divided into four equal-sized domains which are overlapped, as illustrated in Fig. 1, where the symbol Ω_i denotes the domain number i, and $\Gamma_{i,j}$ is the part of the border of Ω_i that penetrates Ω_j. Specific initial conditions are the Dirichlet data on external boundary and, on the entire surface, unit value for reactant and inverse exponential function centered at the temperature ignition point. The solution on the different domains

Ω_i is computed concurrently. Then, the borders are interchanged, which means that every border $\Gamma_{i,j}$ is updated with the current solution on Ω_j. The process is repeated iteratively until the borders remain unchanged in two iterations (i.e. $\|\Gamma_{i,j}^k - \Gamma_{i,j}^{k-1}\|_\infty < 10^{-15}$). Then, the next time step can be computed.

In addition to the equation definition, other requirements are considered. Due to the steepness and high-curvature of the solution (see Fig. 1) a large number of grid points is needed for an accurate result. However, it is a fact that the solution is virtually constant in the areas which have not yet been reached by the wave front, so it can be considered a waste of computational power to use many grid points on such areas (also for the areas the wave leaves). As an alternative, grid-adaptive procedures which dynamically concentrate more grid points where they are needed can be used. In this application, when the traveling front reaches a domain Ω_i, its number of grid points is duplicated. Cubic spline interpolation on the $x-$ axis is used to calculate initial values for the new unknowns.

3.2 Application Design

The application features four scientific components, named `Solve1` to `Solve4`, each one computing the solution on a single domain Ω_i. The SCs use internal data-parallel *red-black* iterations to solve the domain, so every SC itself is a parallel task that divides its domain data to be computed on various processors.

```
PROGRAM reaction-diffusion
  integer :: y0=1, y1=100, d1l=1, d1r=200, d2l=181, d2r=380,// ...
  complex, DOMAIN2D :: omega1/d1l,y0,d1r,y1/
                       omega2/d2l,y0,d2r,y1/
                       // ...
  STRUCTURE
    MULTIBLOCK reaction-mb
      Solve1(omega1) ON PROCS(4),
      Solve2(omega2) ON PROCS(4),
      // ...
    WITH BORDERS
      omega1(d1r,y0,d1r,y1) <- omega2(_)
      omega2(d2l,y0,d2l,y1) <- omega1(_)
      // ...
END
```

```
CONFIGURATION INTERFACE Solve1
  complex, INOUT, DOMAIN2D :: d
  DISTRIBUTE d(BLOCK,*)
END
// ...
```

Fig. 2. Program structure and multiblock (left). Configuration inferface (right).

The structure of the main program in AspectSBASCO is based on a *multiblock* skeleton declaration, as shown in Fig. 2. Once the extension of the domains (i.e. two-dimensional arrays of complex numbers) is defined, the participant SCs, the domains assigned to them and, finally, the communication of borders are indicated using a specific syntax. As an example, the first line of `WITH BORDERS` means that the part of `omega1` delimited by the points `(d1r,y0)` and `(d1r,y1)` will be updated at each iteration by the part of `omega2` delimited by the same points. In this program, it is expressed that every SC executes using four processors. The right side of the picture depicts the configuration interface of `Solve1`. Interfaces for the rest of the SCs are similar. The sole component argument is a two-dimensional domain which, in this case, is distributed by rows.

It should be noted that the border interchange operation can involve complex communication and synchronization among parallel tasks, where the number of processors, data distribution and border extensions need to be considered. By using the multiblock skeleton, the developer is abstracted from these details. She/he only invokes two primitives, `get_data()` and `put_data()`, for receiving and sending border data, respectively, while the system performs efficient communication exploiting the information declared in the skeleton.

Although the procedure described in the last paragraph of Section 3.1 enables better (more accurate) numerical solution, its use may negatively affect the performance. Let us assume that one of the domains increases its size in our executing parallel application. The SC processing such a domain will take longer (in comparison with the other SCs) to finish its task. In addition, the communication of borders (operation repeated several times per iteration) represents a synchronization point for the entire application. For this reason, parallel domain decomposition programs must maintain a similar computation load for every domain, otherwise processors of the domains computed faster will be idle while waiting for new border data. In order to overcome this shortcoming, our application carries out processor re-mapping at run-time. This functionality is implemented using pluggable aspect components, as described later.

The next step entails describing the layer of aspect interfaces which enables the composition and execution of aspects. This application features a total of six aspect components to encapsulate different functionality. Fig. 3 (left) shows component declarations indicating the provided and used interfaces. Interfaces themselves are described in the same figure (right). The four SCs are declared as subclasses of a component root named `Sc` which has common declarations. The idea is that the SCs, influenced by connector declarations (ACNs), will invoke methods on the ACs in order to execute aspect code at different points of the control flow. In this application, the ACNs exploit a set of join points which are predefined in AspectSBASCO and refer to static points (in *multiblock* programs) such as domain initialization, computation of a new time step, evaluation of a new iteration (in current time step), border communication, variation in the number of processors of a SC, and estimation of convergence result.

As can be observed in Fig. 3, aspect component operations usually define their first argument as a reference to the caller SC. The AC implementation can use this reference to invoke operations on the caller component, if necessary. The next paragraphs provide brief component and interface descriptions.

Using `IAccess`, the ACs can modify the state of the SCs. The `Domain2D` object which hosts the solution on the current and the previous time steps can be accessed. The input and output borders, as well as some parameters (e.g. physical system parameters, convergence result), can be manipulated. An AC can modify the domains using `changeDomain()`. The object of type `MapResult` encapsulates the current processor mapping scheme for this application.

Component `LinearSolver` implements `ISolver` and calculates the solution of a linear system $A\mathbf{x} = \mathbf{b}$. At the beginning of each time step, the vector $\mathbf{b}$ is adjusted using `evalFixedTerms()`. Then, the solution $\mathbf{x}$ is obtained calling

```
// COMPONENT DECLARATIONS

component Sc {
  provides IAccess access;
  uses ISolver solver;
  uses IConvergence conver;
  uses IStorage storage;
  uses IAdaptor adaptor;
  uses IMapping mapping;
  uses IState state;
};

component Solve1 : Sc {};
component Solve2 : Sc {};
component Solve3 : Sc {};
component Solve4 : Sc {};

component LinearSolver {
  provides ISolver solver;
  uses IAccess access;
};

component BorderTest {
  provides IConvergence conver;
  uses IAccess access;
};

component Store {
  provides IStorage store;
  uses IAccess access;
};

component GridAdaptor {
  provides IAdaptor adaptor;
  uses IAccess access;
};

component Scheduler {
  provides IMapping mapping;
  uses IAccess access;
};

component State {
  provides IState state;
  uses IAccess access;
};
```

```
// ASPECT INTERFACE DECLARATIONS

interface IAccess {
  Domain2D getDomain(in unsigned currentOrPrevious);
  sequence<Domain2D> getBorders(in unsigned inputOrOutput);
  double getParam(in string param_name);
  void setParam(in string param_name, in double value);
  void changeDomain(in Domain2D currDom, in Domain2D prevDom);
  MapResult getMapResult();
  void setMapResult(in SchResult schRes);
};
interface ISolver {
  void setup(in Sc sc, in Domain2D currDom);
  void evalFixedTerms(in Sc sc, in Domain2D prevDom);
  void eval(in Sc sc, in Domain2D currDom, in Domain2D prevDom);
  void checkDomain(in Sc sc, in Domain2D currDom, in Domain2D prevDom);
  void checkMapResult(in Sc sc, in MapResult mapRes);
};
interface IConvergence {
  void setup(in Sc sc, in sequence<Domain2D> inputBorders);
  void copy(in Sc sc, in sequence<Domain2D> inputBorders)
  boolean evaluate(in Sc sc, in sequence<Domain2D> inputBorders);
  void checkMapResult(in Sc sc, in MapResult mapRes,
                      in sequence<Domain2D> inputBorders);
};
interface IStorage {
  void load(in Sc sc, in Domain2D currDom);
  void save(in Sc sc, in Domain2D currDom);
};
interface IAdaptor {
  void setup(in Sc sc, in Domain2D currDom, in Domain2D prevDom);
  void adapt(in Sc sc, in Domain2D currDom);
  void initPut(in Sc sc);
  void finishGet(in Sc sc);
  void communicateGridMode(in Sc sc, in MapResult mapRes);
  void finishApp(in Sc sc, in Domain2D currDom);
};
interface IMapping {
  void setup(in Sc sc);
  void startTime(in Sc sc);
  void stopTime(in Sc sc);
  void remap(in Sc sc);
  void communicateMapResult(in Sc sc, in MapResult mapRes);
};
interface IState {
  void initRemap(in Sc sc, in Domain2D currDom, in MapResult mapRes)
  void finishRemap(in Sc sc, in Domain2D currDom, in MapResult mapRes)
};
```

Fig. 3. Component and aspect interfaces used in the reaction-diffusion problem

`eval()` iteratively. The functions `checkDomain()` and `checkMapResult()` are executed, respectively, when the domain is replaced and when the mapping of processors changes. These functions adapt the component to the new settings.

Component `BorderTest` calculates the convergence of the method. Each time a SC invokes `get_data()` to receive new borders, the code of `copy()` stores the values of previously used borders. The function `evaluate()` compares successive border data to determine if the calculation of a time step must stop.

Component `Store` implements a simple persistence characteristic. The execution time of this application can range (from several minutes to dozens of hours in modest parallel computers) simply changing the domain size and convergence criteria. `Store` enables the final distributed numerical result to be the initial condition of a subsequent execution. This way a simulation can be completed in several sessions. The functions `load()` and `save()` manage the serialization.

The grid adaptation procedure is encapsulated in `GridAdaptor`, which implements `IAdaptor`. The method named `adapt()` changes both the current domain and the $x-$ space step length. This means either increasing or decreasing the number of grid points in function of the variable v. The component uses cubic splines to interpolate the new points in the $x-$ axis. In addition, this

```
ACN adaptor_acn on component Sc {
  advice before on put_data_call {
    getAdaptor()->initPut(this);
  };
  advice after on resize_call {
    getAdaptor()->communicateGridMode(this, getMapResult());
  };
  advice after on init_call {
    getAdaptor()->setup(this, getDomain(0), getDomain(1));
  };
  advice before on step_call {
    getAdaptor()->adapt(this, getDomain(0));
  };
  // ...
};
```

Fig. 4. Aspect connector declaration for grid-adaptive procedure management

component manages the borders in accordance with the type of domain being used. For instance, each time borders are received, the function `finishGet()` adapts received data considering the real extension of the new domain.

Component `Scheduler` determines how processors should be re-distributed for a better computation balance. The methods `startTime()` and `stopTime()` are called just before and after the code blocks which will be monitored. These functions allow `Scheduler` to calculate the execution time of the SCs. When the function `remap()` is invoked, `Scheduler` consults time information to establish a better processor mapping. The algorithm used in this application is quite straightforward and it consists of comparing two SCs which are the components having the highest and slowest execution time. If the difference is greater than a threshold the "fast" SC releases one processor which is taken by the "slow" SC.

Finally, `State` is in charge of communicating the computation state in processor re-mapping operations. For instance, if a SC acquires one more processor, the current domain data has to be re-distributed among a higher number of processors. Other internal variables have also to be communicated. The methods `initRemap()` and `finishRemap()` implement the state communication and are called, respectively, just before and after any change in the scheme of processors.

The execution of aspect code is governed by the so-called aspect connectors (ACNs). An ACN is defined on (i.e. affects) a group of SCs and contains one or several *advice declarations*. Every advice consists of an *advice header* indicating a combination of join points at which the *advice body* will be executed. The syntax chosen for advice body is C++. The only code statements allowed here are invocations to the AC operations (i.e. interactions). Advice header usually indicates the instant at which advice body is executed. Allowed values are *before* and *after* the corresponding join point. In Fig. 4, part of an ACN declaration for the execution of `GridAdaptor` is shown. For instance, the first advice means that just before border communication routine `put_data()` is executed on any SC, the operation `initPut()` on `GridAdaptor` is invoked. The second advice expresses that `communicateGridMode()` is invoked on the same AC after the number of processors of any participant SC changes. A reference to the component `GridAdaptor` is retrieved using `getAdaptor()`. Similar ACN structures are needed for the management of the other ACs. Although in this example the use of predefined join points was enough, advice headers can refer to methods on the aspect interface as alternative join points. AspectSBASCO provides mechanisms to express the precedence of a set of advices affecting the same join point.

3.3 Implementation Issues

The current implementation of AspectSBASCO consists of a runtime system based on MPI, a programming framework in C++ and a source-to-source compiler to produce target language code structures. Scientific and aspect components are implemented using C++ and MPI. SCs inherit from the class `ScRoot` which provides access to distributed arguments. This class declares `get_data()` and `put_data()` as member functions. ACs are C++ classes that implement all operations of the corresponding aspect interfaces. An instance of the class `Framework` providing a set of common services can be retrieved at any point in the application. For example, `Framework` has methods that return up-to-date MPI intra- and inter-communicators to implement component parallelism.

An application consists of a collection of MPI programs. Specifically, one MPI program (so-called *worker*) is associated with each SC. Each worker manages one single SC instance together with a set of AC instances. The program executes using a group of processes whose size was expressed in the high-level composition language. When an aspect affects different SCs, the corresponding AC appears in several workers, and so aspect instances are created in several groups of processes.

Aspect weaving is the mechanism by which aspects and base code are combined to produce final applications. In our approach, this process is carried out at compile-time by modifying the source code of the SCs. More specifically, the aspect weaver examines ACN structures in order to include advice body interaction code at the corresponding points indicated in connector declarations. Every time a SC executes AC functionality (as a result of using the ACNs) the method invocation is carried out (in parallel) in all processes that contain both component instances. Then, it is the responsibility of the aspect developer to implement functionality that may range from simple local processing to complex parallel computation involving several groups of processes. The important thing is that our approach avoids any type of complex runtime structure to support aspect execution. The overhead of invoking ACs in final applications is equivalent to the cost of a standard C++ method call (the one declared in advice body).

Communication based on `get_data()` and `put_data()` (namely border interchange in this case study) is based on point-to-point message passing managed by the runtime system. This communication is efficiently carried out due to the information expressed in component configuration interfaces and skeletons.

4 Conclusions

This paper describes the use of AspectSBASCO for the development of a reaction-diffusion parallel problem. The complexity of functional and non-functional application concerns is managed effectively by means of a combination of paradigms. The resulting approach leads to improved interoperability and application evolvability. In this case study, heterogeneous concerns such as grid adaptation and processor re-mapping procedures are implemented as well-modularized aspects. Pluggability is also provided. For instance, simply plugging the component that manages processor re-mapping, the application achieves the same numerical result more efficiently due to a better balance of the computation load. In addition, using high-level parallel skeletons raises the abstraction level of the development.

References

1. Armstrong, R., et al.: The CCA Component Model for High Performance Scientific Computing. Concurrency and Computation: Practice and Experience 18(2), 215–229 (2006)
2. Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Flexible Skeletal Programming with eSkel. In: Proc. of the 11th International Euro-Par Conference, Lisboa, Portugal, pp. 761–770 (2005)
3. Danelutto, M.: QoS in Parallel Programming Through Application Managers. In: Proc. of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Lugano, Switzerland, pp. 282–289 (2005)
4. Díaz, M., Rubio, B., Soler, E., Troya, J.M.: SBASCO: Skeleton-based Scientific Components. In: Proc. of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, A Coruña, Spain, pp. 318–324 (2004)
5. Díaz, M., et al.: Using SBASCO to Solve Reaction-Diffusion Equations in Two-Dimensional Irregular Domains. In: Proc. of the 3rd International Workshop on Practical Aspects of Parallel Programming, Reading, UK, pp. 912–919 (2006)
6. Díaz, M., et al.: Adding Aspect-Oriented Concepts to the High-Performance Component Model of SBASCO. In: Proc. of the 17th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Weimar, Germany (to appear, 2009) (The paper can be accessed), http://www.lcc.uma.es/~tolo/publications.html
7. Harbulot, B., Gurd, J.: A Join Point for Loops in AspectJ. In: Proc. of the 5th International Conference on Aspect-Oriented Sofware Development, Lancaster, UK, pp. 122–131 (2006)
8. Heineman, G.T., Council, W.T.: Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley, Reading (2001)
9. Kiczales, G., et al.: Aspect-Oriented Programming. In: Proc. of the European Conference on Object-Oriented Programming, Jyvskyl, Finland, pp. 220–242 (1997)
10. Pelagatti, S.: Structured Development of Parallel Programs. Taylor & Francis, Abington (1998)
11. Quarteroni, A., Valli, A.: Domain Decomposition for Partial Differential Equations. Oxford Science Publications (1999)
12. Sobral, J.L.: Incrementally Developing Parallel Applications with AspectJ. In: Proc. of the 20th International Parallel and Distributed Processing Symposium, Rohdes, Greece, p. 10 (2006)
13. Suvée, D., Fraine, B., Vanderperren, W.: A Symmetric and Unified Approach Towards Combining Aspect-Oriented and Component-Based Software Development. In: Proc. of the 9th International SIGSOFT Symposium on Component-Based Software Engineering, Stockholm, Sweeden, pp. 114–122 (2006)
14. Vanneschi, M.: The Programming Model of ASSIST, an Environment for Parallel and Distributed Portable Applications. Parallel Computing 28(12), 1709–1732 (2002)

Balancing Scientist Needs and Volunteer Preferences in Volunteer Computing Using Constraint Optimization

James Atlas, Trilce Estrada, Keith Decker, and Michela Taufer

University of Delaware, Newark, DE 19716 U.S.A.
{atlas,estrada,decker,taufer}@cis.udel.edu

Abstract. BOINC is a middleware for Volunteer Computing. In BOINC projects, heterogeneous resources distributed across the Internet are used for large-scale scientific simulations. The large need for resources in BOINC projects often competes with volunteer preferences: volunteers can impose limits on the use of their idle resources. Most of the time, maximum project performance can be achieved only when volunteer preferences are neglected.

To address this problem, we propose a novel optimization procedure based on constraint optimization techniques that actively allocates volunteer resources to improve project throughput and, at the same time, aims to preserve volunteer preferences. We show the increase in project throughput obtained with our approach and discuss the trade-off between volunteer preferences and project throughput.

Keywords: Volunteer Computing, Constraint Optimization.

1 Introduction

Volunteer Computing (VC) is a form of distributed computing in which ordinary people (i.e., volunteers) volunteer processing and storage resources to computing projects. BOINC is a well-known middleware for VC [1] supporting scientific computing projects (e.g., physics, biology, and medicine). The main strength of BOINC systems is its capability to provide scientists with PetaFLOPs of computing power at low cost. The VC community powered by BOINC currently counts approximately 50 projects and 580 000 volunteer computers supplying an average of 1.2 PetaFLOPs to these projects.

VC resources increasingly include diverse platforms such as video game consoles (Playstations) and graphics processing units (GPUs). Some VC projects are able to customize their code to benefit from performance features of these platforms. This creates an instance of a general resource allocation problem where jobs have disparate performance profiles depending on the platform of execution. In addition, volunteers can specify resource allocation preferences over a subset of VC projects that they want to participate in, essentially constraining the possible jobs the server can allocate to a volunteer host and ultimately hinders the maximum throughput of projects. On the other hand, ignoring volunteer's preferences for performance sake can upset the donors who can withdraw their resources. Server scheduling policies decide which jobs to assign to volunteer hosts given a set of unallocated jobs and volunteer preferences. Ideally these policies should optimize allocation of jobs across heterogeneous volunteer resources and, at the same time, preserve volunteer preferences in the best way and with the highest performance. This task is made more challenging by the increasing heterogeneity of VC systems.

To address this challenge, we propose a novel optimization procedure that actively allocates volunteer resources to improve project throughput and preserve volunteer preferences. Our optimization procedure is based on constraint optimization techniques (COP) and provides a robust framework for maximizing the contributions of diverse resources. We evaluate our approach against the current, most advanced allocation strategies of BOINC using EmBOINC, a full-scale emulation of the BOINC platform using realistic trace populations of volunteer hosts (including heterogeneity, churn, availability, reliability). This paper shows the increase in project throughput obtained with our approach and discusses the trade-off between volunteer preferences and project throughput.

This paper is organized as follows: Section 2 presents a short overview of important background concepts such as VC, BOINC, our emulation of BOINC projects, and COP. In Section 3 we introduce our optimization procedure. Section 4 compares our approach with the current practice scheduling policies of BOINC. Section 5 concludes the paper and presents some future work.

2 Background and Related Work

2.1 Volunteer Computing

Volunteer Computing (VC) projects employ computing resources (e.g., desktops, notebooks, and servers) owned by ordinary people and connected to the Internet. Traditionally, VC projects target large search problems in science and, therefore, generate large sets of jobs that are distributed across VC resources. Replication of jobs is used to address the volatility of these systems as well as other issues like malicious attacks, hardware malfunctions, or software modifications that ultimately affect the reliability of results. Replicas of jobs (job instances) are distributed to different VC resources (hosts) that execute them. When finished, the hosts send their results to the project server, which collects the results and distinguishes between successful and unsuccessful results. Unsuccessful results are those that either are erroneous or are returned too late, i.e., timed-out.

2.2 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [1] is an open-source system that harnesses the computing power and storage capacity of thousands or millions of PCs owned by volunteers for scientific simulations. The computing resources available to a BOINC project are highly diverse: the hosts differ by orders of magnitude in their processor speed, available RAM, disk space, and network connection speed. Recently, the heterogeneity of BOINC platforms has been enriched by the adding of GPUs and Playstations. The BOINC model involves projects and volunteers. Projects are organizations (typically academic research groups) that need computing power. Projects are independent and have different resource requirements. Volunteers participate by running the BOINC client software on their computers (hosts). Volunteers can attach their hosts to one or multiple projects (preferred projects). When a BOINC client is attached to a project, it periodically issues a *request* to the project's server. The request includes a description of the host and its current workload, descriptions of recently-completed jobs, and a request for new jobs based on the volunteer's preferences. The *reply* from the server may contain a set of new jobs. Multiple job results may be returned; this reduces the rate of scheduler requests and accommodates clients that are disconnected from the Internet for long periods.

2.3 Scheduling in BOINC

Initially BOINC scheduling policies relied on greedy and naive policies. Recently, more sophisticated server-side scheduling policies have been implemented in several BOINC projects. Currently, World Community Grid has a number of criteria for job assignment [2], based on host and job diversity (e.g., size of the job and speed of the host relative to an estimated statistical distribution, disk and memory requirements for the job to be completed, homogeneous redundancy [3] and host error rate). A scoring-based scheduling policy uses a linear combination of these terms to select the best set of jobs for a given host. Projects can adjust the weights of these terms, or they can replace the scoring function entirely. None of these policies search for trade-off between volunteer preferences and project requirements.

2.4 Emulating BOINC

The scheduling policies embedded in the BOINC server have a large impact on the project throughput and other performance metrics. Unfortunately, it is difficult (if not impossible) to do controlled performance experiments in the context of a large VC project because there are many factors that cannot be controlled and because poorly-performing mechanisms can waste volunteer resources. However, exploring new policies can be done in simulated environments, where it is possible to test a wider range of hypotheses in a shorter period of time without affecting the BOINC community. To evaluate our scheduling approach we use EmBOINC (Emulator of BOINC Projects), a trace-driven emulator that models heterogeneous hosts and their interaction with a real BOINC server [4]. By plugging into a BOINC server, EmBOINC triggers the server's daemons to generate and distribute jobs to the EmBOINC hosts. EmBOINC uses statistical information obtained from real BOINC traces to characterize volatile, heterogeneous, and error-prone hosts. As it occurs in real BOINC projects, EmBOINC can emulate different projects running simultaneously. Projects can share the simulated hosts partially or completely. For every project, the associated hosts can have different levels of heterogeneity, errors, availability, and reliability. Using EmBOINC, different patterns for job generations as well as different policies for job distribution and validation can be studied.

2.5 Constraint Optimization

Many historical problems in the AI community can be transformed into Constraint Satisfaction Problems (CSP). Early domains for constraint satisfaction problems included job shop scheduling [5] and resource allocation [6]. Many of these domains involve overly constrained problems that are difficult or impossible to satisfy for every constraint. Recent approaches to solving problems in these domains rely on optimization techniques that map constraints into multi-valued utility functions. Instead of finding an assignment that satisfies all constraints, these approaches find an assignment that produces a high level of global utility. A typical constraint optimization problem (COP) begins with a constraint graph mapping of a problem. The COP mapping is defined as a set of n variables and m constraints producing the tuple $< X, D, U >$ where:

- $X = \{x_1,..,x_n\}$ is a set of variables, each one assigned to a unique agent
- $D = \{d_1,..,d_n\}$ is a set of finite domains for each variable
- $U = \{u_1,..,u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility for each combination of values among these variables

An optimal solution to a COP instance consists of an assignment of values in D to X such that the sum of utilities in U is maximal. An example COP instance for the standard graph coloring problem with weighted utilities is shown in Figure 1.

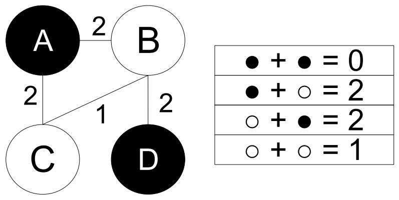


Fig. 1. COP Example: Simple graph coloring problem with utility functions. Coloring shown is optimal for this problem, and utility values are next to the constraints.

3 Methodology

The idea behind our approach is that we can increase throughput for BOINC projects by intelligently coordinating schedules for volunteers. To achieve this, we develop a constraint optimization (COP) mapping that pursues high throughput while trying to adhere to the volunteer's preferences.

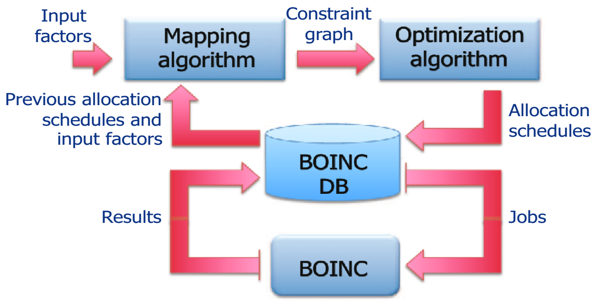


Fig. 2. Overview of our constraint optimization approach

3.1 Approach Overview

In Figure 2 we show the different components of our solution to optimize BOINC schedules. The process can be run continuously in a real-time environment because it only interacts with the BOINC server through its database. Thus, the flow through the diagram can be considered a cycle, beginning with the input factors and ending with an update to the BOINC database. The input factors are passed to the mapping layer (mapping application) to convert the actual BOINC scheduling data and parameters into a constraint graph. The constraint graph is used as a general representation for a COP and is passed into the optimization algorithm. The optimization algorithm determines a new containing allocation schedules for each volunteer's resources to the BOINC projects. These allocation schedules are then updated in the database and are used to determine which jobs to return when a volunteer requests new work.

3.2 Input Factors

The first step in our approach is to define input factors that we will consider for the optimization. The value of each factor directly affects the outcome of the optimization procedure. A list of these factors and the actor responsible for providing their value appears in Table 1. In addition to these definable factors, input values are also derived from the BOINC database about volunteer resources (e.g., estimated flops) and project characteristics (e.g., GPU/coprocessor support).

3.3 Mapping BOINC Schedules to COP

The next step is to map the input factors into a coherent problem representation. We use a constraint optimization representation, so we will provide a mapping from the input factors to a constraint graph. A constraint graph contains variables as nodes and constraints as edges. We consider two types of variable nodes:

Allocation Schedule (AS) represents the allocated volunteer schedule. The AS factor listed earlier determines the starting value for this variable. Possible variable assignments represent new allocation schedules for the volunteer.

Table 1. Input factors

ID	Factor	Actor	Description
VP	Volunteer Preferences	Volunteer	Set of projects preferences for a volunteer. Volunteers assign 1 to preferred projects and 0 otherwise.
PTN	Project Throughput Need	Scientist	A target number of results returned per project per day. Specified in GFLOPS/s or by job deadline.
AS	Allocation Schedule	BOINC-COP	The specific allocation of volunteer resources to different projects. The current set of allocation schedules is input and a new set of allocation schedules is output. Percent per project per volunteer.
TvP	Throughput vs. Preference	Scientist	A weight between 0 and 1 for favoring project throughput over volunteer preferences, where 0 means full matching of volunteer preferences and 1 complete ignore.

Project Throughput Need (PTN) represents a level between 0 and 1 to which the project needs additional volunteer resources. In relation to the TN factor listed earlier, a value of 1 means that the project needs additional resources and is currently short of its target. A value of 0 means that the project has no use for additional resources and has already met its throughput target. A value in between means that a project has met its throughput target but could use additional resources.

Each volunteer has one AS variable and each project has one PTN variable. We now create binary constraints between each volunteer (AS variable) and every project (PTN variable) the volunteer is willing to work for. This constraint returns a utility value that represents the utility of a volunteer's current allocation schedule for a given project's level of throughput need. The value of this constraint, $U(N, M)$ for project N and volunteer M is:

$$\begin{aligned} U(N,M) = {} & P_N(AS_M) \cdot PTN_N \cdot C_M(N) \\ & \cdot (W_{AT} + (1 - W_{AT}) \cdot VP_M(N)) \end{aligned} \tag{1}$$

Where:

– $P_N(AS_M)$ is the percent allocated to project N in the schedule of M
– PTN_N is the level of throughput need for project N (between 0 and 1)
– $C_M(N)$ is the contribution of volunteer M to project N in GFLOP/s
– W_{AT} is the weight for optimizing project throughput (between 0 and 1)
– $VP_M(N)$ is volunteer M's preference for project N (between 0 and 1)

These variables and constraints form the constraint graph representation of our original input factors. We now take this general COP representation and apply our optimization algorithm to find a high utility scheduling policy.

3.4 Optimization Algorithm

Our optimization algorithm takes as input the constraint graph formed in the previous section, and solves for a new scheduling policy containing allocations that optimize each volunteer's resources to the BOINC projects. Our algorithm is a modified version of the stochastic gain algorithm described in [7]. We add support for derived variables and random gain delays. The following steps are performed in parallel for each variable:

1. Get max local gain (best schedule for current state of PTN variables)
2. With probability p, change to max assignment (new value for AS variable; setting p too high can prevent full convergence)
3. Derive new values from neighbors (for PTN variables from all schedules)

These three local search steps are performed for a number of cycles; at a specified maximum amount of time the algorithm is terminated and the best utility assignment encountered so far is chosen. The algorithm converges with low enough p. We implemented a delay in number of cycles between changes to the same AS variable to also help with convergence. The algorithm can scale to tens of thousands of variables. If we require optimization of larger sets of volunteers, we can pre-process the set and cluster volunteers into groups that share similar preferences and resource contribution characteristics. Then we simply treat each group as a super-volunteer with one volunteer variable which represents an identical allocation schedule assigned to all volunteers in the group.

3.5 Integration with BOINC Server

The output from the optimization algorithm is a set of allocation schedules for each volunteer. We store these allocation schedules in the BOINC database. Modifying the BOINC server to use our allocation schedules is easily done. The BOINC scheduler uses a scoring mechanism to handle each request for work from a volunteer. Each unassigned job receives a score for possible assignment to the requesting volunteer. The highest scoring set of jobs that fill the amount of time requested by the volunteer are sent. To integrate our allocation schedules we simply add a value to the score if the job matches the volunteer's allocation schedule. The schedule contains a number between 0 and 1 for each project for this volunteer. We generate a random number between 0 and 1 and if it is less than the schedule allocation number than that job receives a higher score. Thus, over time the allocation of jobs will match the percentages specified in the schedule. Typically the optimization process, run in parallel with the BOINC scheduler, took less than one minute to complete. Large systems can tune the frequency of optimization, or obtain faster, approximate optimizations.

4 Evaluation

To evaluate our approach, we use EmBOINC and consider different scenarios matching the behavior of real BOINC projects.

4.1 Performance Metrics

With EmBOINC it is possible to conduct extensive experiments targeting different performance metrics. We measure the following metrics:

Throughput-based metrics include *project throughput* in terms of total results returned to the scientist or results returned per day.

Preference metrics count the total number of jobs executed by volunteers for preferred and non-preferred projects.

Deadline-based metrics count the number of job results that completed prior to the deadline given by the scientist.

4.2 Scenarios

We tested our approach in two orthogonal scenarios with respect to the way jobs are generated. We considered three projects running simultaneously.

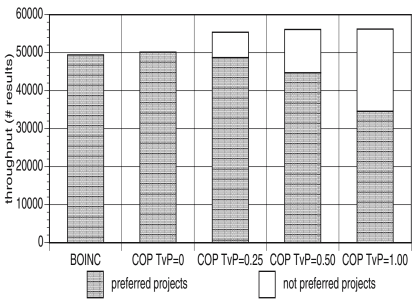
Scenario 1. In this scenario we tested the ability of our solution to optimize throughput with *uniform generation of jobs*. Project 1 (uses only CPU) and 3 (uses CPU or GPU) were given a target of 20% of the overall throughput each. Project 2 (uses only CPU) targeted 60% of the throughput. The scenario generated 4000 jobs for Projects 1 and 3 and 12000 for Project 2. This scenario is typical for projects without deadlines like SETI@Home (http://setiathome.berkeley.edu).

Scenario 2. In this scenario we tested the ability of our solution to optimize throughput with *irregular generation of jobs*. For Project 2 we randomly injected 8 batches of 1500 jobs over 25 days, with a 10 day deadline for each batch. For Projects 1 and 3 we kept a uniform generation of 4000 jobs each. This scenario is similar to the real-world case in Critical Assessment of techniques for protein Structure Prediction (CASP). During the biennial CASP competition (http://predictioncenter.org/), new targets (amino acid sequences) are released to the participants almost every day with a deadline of 15 days for the target 3D prediction. Projects such as Predictor@Home and Rosetta@Home belong to this class of scenarios.

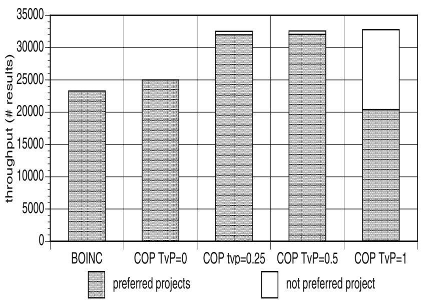
We ran our simulations using a fixed amount of simulated time (25 days). We used a base set of 500 volunteers, of which 20% have GPUs. Each volunteer randomly chose 1 or 2 preferred projects of the 3 possible. All other host characteristics (e.g., CPU speed, memory) were randomly generated based on trace data from real-world BOINC projects.

4.3 Results

Figure 3(a) shows total throughput (number of results) for the three projects in Scenario 1 using BOINC and different levels of TvP for our constraint optimization (COP) approach. A low TvP means closer matching to volunteer preferences, where TvP equal to zero means perfect matching. The gray bar is jobs executed



(a) Scenario 1



(b) Scenario 2

Fig. 3. Throughput (number of results) for the two different scenarios

by volunteers for their preferred projects, and the white bar is jobs executed for non-preferred projects. If we fully comply with volunteer preferences (TvP=0), our system has the same performance as BOINC. As we increase the TvP factor, we increase the total throughput of the system, but we diverge from volunteer preferences. A complete violation of volunteer preferences is not needed to achieve high levels of throughput, as TvP setting of 0.25 already achieves higher throughput than BOINC (+12%). Figure 3(b) shows the same metric for the project with deadlines in Scenario 2. In this scenario, batches of jobs are randomly injected, with a 10 day deadline for each batch. Throughput is the number of results for Project 2 returned to the BOINC server before their deadline. As in Scenario 1, the gray bar is jobs executed by volunteers for their preferred projects, and the white bar is jobs executed for non-preferred projects. If we fully comply with volunteer preferences (TvP=0), our system performs slightly better than BOINC (+7.4%). As we increase the TvP factor, the throughput increases significantly. Again, we do not need to completely violate the volunteer preferences; with a TvP setting of 0.25 we gain 39.6% throughput. Increasing the TvP factor allows our system to re-allocate volunteer resources to jobs with upcoming deadlines. Note that the percentage of jobs performed for non-preferred projects is minimal (1.6%).

Overall, we see that our approach increases throughput for both uniform and irregular job generation scenarios. Increasing the TvP factor allows us to achieve higher throughput at a cost of making volunteers execute jobs for non-preferred projects. A TvP trade-off of 0.25 provides maximum throughput with minimum volunteer preference violation. The reason for increase in total throughput is *not* due to additional idle cycles (we had the same resource idle time for the runs); it is because we use the resources more efficiently. For both approaches we made sure that there were new jobs available for every volunteer request. The results were validated by repeating each simulation three times with the same observed behavior.

5 Conclusions and Future Work

We presented a novel optimization procedure based on constraint optimization techniques that actively allocates volunteer resources to improve project throughput and, at the same time, aims to preserve volunteer preferences. We show two scenarios that exhibit increased project throughput (up to 39.6%) for a minimal trade-off in execution of jobs for non-preferred projects. Our results show that it is possible to balance the needs of scientists with the preferences of volunteers in VC projects. In the future, we intend to extend our approach to optimize credit given to volunteers. Currently, volunteer credit is based on the number of FLOPS executed by the volunteer. However, in some scenarios scientists may want to assign greater credit per FLOP for one project than another. In this case the volunteer would want to optimize the amount of credit they earn. This is similar to scenarios in grid and cloud computing, and we intend to examine how our approach can be applied to these related problems.

Acknowledgment

This work was supported by the National Science Foundation, grant #OCI-080265, DAPLDS - a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling.

References

1. Anderson, D.P.: BOINC: A System for Public-Resource Computing and Storage. In: Proc. of the 5th IEEE/ACM International Workshop on Grid Computing (2004)
2. Anderson, D.P., Reed, K.: Celebrating Diversity in Volunteer Computing. In: Proc. of the Hawaii International Conference on System Sciences (HICSS) (2009)
3. Taufer, M., Anderson, D., Cicotti, P., Brooks III, C.L.: Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing. In: Proc. of the 14th Heterogeneous Computing Workshop (2005)
4. Estrada, T., Reed, K., Anderson, D., Taufer, M.: Emboinc: An emulator for performance analysis of boinc projects. In: Proc. of PCGrid (May 2009)
5. Liu, J., Sycara, K.P.: Exploiting problem structure for distributed constraint optimization. In: ICMAS 1995, pp. 246–254 (1995)
6. Modi, P.J., Jung, H., Tambe, M., Shen, W.-M., Kulkarni, S.: A dynamic distributed constraint satisfaction approach to resource allocation. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 685–700. Springer, Heidelberg (2001)
7. Zhang, W., Xing, Z., Wang, G., Wittenburg, L.: An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In: AAMAS 2003 (2003)

Scheduling and Load Balancing

Hiding Communication Latency with Non-SPMD, Graph-Based Execution

Jacob Sorensen and Scott B. Baden

Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0404 USA
http://www.cse.ucsd.edu/groups/hpcl/scg

Abstract. Reformulating an algorithm to mask communication delays is crucial in maintaining scalability, but traditional solutions embed the overlap strategy into the application. We present an alternative approach based on dataflow, that factors the overlap strategy out of the application. Using this approach we are able to reduce communication delays, meeting and in many cases exceeding performance obtained with traditional hand coded applications.

Keywords: parallel programming, latency tolerance, non-SPMD, coarse grain dataflow.

1 Introduction

Spurred on by the multi-core processor, scalable systems have the potential to enable simulations of remarkable fidelity and complexity, leading to new scientific discovery. However, improvements in processor performance amplify the cost of off-chip data motion, and applications must cope by with this trend by tolerating latency. Implementing and tuning an application to overlap communication with computation is daunting for the domain scientist, and a challenge even for the expert programmer. Traditionally, the overlap strategy is embedded into the application, and relies on split phase coding. Software development is tedious and prone to error, and the application suffers from non-robust performance.

Overlap strategies expose opportunities to mask communication costs by relaxing the total ordering imposed by traditional bulk synchronous implementation, e.g. with MPI [1]. A compiler may be able to determine a suitable partial ordering in some cases, but often the partial orderings are difficult to analyze, even by hand.

A natural way to realize partial orderings is by means of a task precedence graph, or *task graph* for short. Once the program has been expressed in terms of a task graph, a scheduler executes the partially ordered tasks according to the flow of data, e.g. dataflow [2,3,4,5], realizing overlap automatically. We have implemented this approach with a run time library, with run time services to support the data flow semantics via background threads. These services reorder communication and computation tasks dynamically according to the flow of information, automatically choreographing communication to move it out of the critical path of computation. We have implemented applications using this approach, enabling us to factor policy and scheduling decisions out of the application code.

This paper makes two contributions. First, we show that a task precedence graph formulation is able to support latency tolerance, meeting and in many cases exceeding performance of traditional split phase encodings. Moreover, this formulation separates implementation policy from correctness. Second, we show that the approach supports application specific scheduling without affecting correctness of user code. Our task precedence graph supports performance metadata [6] that may be used to improve performance, in particular, to fine tune the scheduler.

2 Task Precedence Graph Representation

Scientific applications spend most of their time executing loop nests. We may describe a loop nest using an *Iteration Space Graph* (ISG), which represents the underlying dependence structure. Any scheme to parallelize an application must preserve the dependencies in the constituent ISGs. We may construct a task precedence graph using this ISG in which each task corresponds to a region of the iteration space. We call the resultant graph a *TaskGraph.* In a classic MPI implementation tasks are usually mapped 1:1 to processors. However, in order to mask data transfer delays, Little's law [7] prescribes that we render many tasks for each processor. As a result, we must solve a scheduling problem. Herein lies the difficulty: classic overlap strategies rely on split phase algorithms that embed overlap strategy into the application source code, resulting in high software development costs, and difficulty in porting code to new hardware.

Alternatively, we may execute the TaskGraph under the dataflow semantics [2,3,4]. Parallelism arises among independent tasks and interdependent tasks are enabled according to the flow of data among them. There is no need to embed scheduling policies into the application since these are handled by background services.

We have constructed a library, called *Thyme*, which enables us to implement applications in this way. Thyme avoids the need to write complicated split phase algorithms and decouples the overlap strategy from application correctness. Although Thyme's API may be used to develop applications directly, we envision that it will ultimately become part of a run time support library for a compiler. Space limitations prevent us from discussing this API, which, together with a detailed presentation of the implementation, will be described elsewhere.

Thyme is currently implemented as a C++ class library on top of MPI and pthreads, and implements two primary datatypes – *Task* and *TaskGraph.* A Thyme program constructs and executes a set of TaskGraphs under the control of the *Run Time Services* which process task completions and arrivals, move data among dependent tasks, and invoke a Scheduler. The services are decentralized, and run on all processing modules, communicating as necessary to manage TaskGraph execution. A TaskGraph is a distributed data structure and it executes according to the owner computes rule. Each task is assigned an owning processing module, but any processor within the module may execute the task, since processors share memory within a node.

The Scheduler maintains a priority queue of tasks from the TaskGraph that are ready but not yet executing. Each task has an associated *priority.* This information exists as performance meta-data [6] decorating the task graph. Meta-data may be be specified by the programmer in order to improve performance, but does not affect correctness since task graph execution preserves all dependence constraints. Thus, the programmer is free to explore application-specific scheduling without reformulating the application. (The Thyme user may substitute their own scheduler in place of the default in order to further tune application performance.)

3 Experiments

We ran on two large-scale systems. *DataStar,* located at the San Diego Supercomputer Center, is an IBM system running AIX 5.2, with 8-way nodes containing 1.5 GHz Power4+ processors with 16 Gigabytes of shared memory, connected by a Federation switch. *Thunder,* located at Lawrence Livermore National Laboratory, is a Linux cluster running CHAOS version 3.3, with a Quadrics QSNET-II interconnect based on Elan4 and Elite4 components. Each "Madison Tiger4" node comprises four Itanium2 CPUs running at 1.4 GHz, with 8 Gigabytes of shared memory. We compiled on DataStar using `mpCC_r`, which invoked version 8.0 of the `xlC_r` compiler. IBM's `ESSL` version 4.2.0.3 provided the high performance matrix multiply routine `dgemm()` and FFTW 3.0 provided the FFT. We compiled on Thunder with `mpiicpc`, which invoked `icpc 9.1`. Intel's M̂KL 8.1.1 provided `dgemm` and FFT (the latter through the FFTW 3.0 interface).

We tested Thyme's ability to achieve overlap with three applications coming from Colella's seven application motifs [8]. Jacobi3D, a 3-D iterative Poisson solver (Dirichlet Boundary Conditions); MMULT, Matrix Multiplication, and the NAS-FT parallel benchmark, ver. 3.0 [9], which is dominated by a 3D Fast Fourier Transform. For each application we compared the Thyme implementation against two variants written with MPI. The *baseline* variant (BASE) uses blocking communication and does not attempt to overlap communication with computation. The *explicit overlap* variant (OLAP) uses asynchronous nonblocking communication to implement a split-phase algorithm to overlap communication with computation. The *Thyme* variant does not make MPI calls, since the run time services handle data motion automatically. These services commandeer one core per node to carry out their activities. Thus, although Thyme uses the same number of processors as non-Thyme variants in our experiments, fewer processors actually perform the computation.

We also report an *IDEAL* running time, which is the time required to perform computational work only. We obtained this time by disabling communication in BASE. Although the computed results are incorrect, the amount of computational work performed is not affected. This allows us to indirectly measure the cost of communication and thus establish an upper bound on the potential for improving performance by masking data motion costs.

Jacobi3D. *Jacobi3D* iteratively updates a 3D mesh using a 7-point stencil. We split the mesh uniformly and employ ghost cells to store border data from (up to) six neighboring sub-domains. The BASE variant used an $8 \times 4 \times 8$ processor geometry which is optimal for 256 processors. Ghost cells are exchanged prior to each iteration using `SendRecv`. The OLAP variant pre-fetches ghost cells [10]. It further subdivides each processor's subdomain into an inner core and an outer annulus. The annulus is a thin shell, one cell thick, encircling the inner core and inscribed inside the ghost region. Unlike the outer annulus, the computation on the inner core does not depend on the ghost cells; it is relaxed simultaneously with ghost cell exchange. Once all the ghost cells have all arrived, the outer annulus is then relaxed. The *Thyme* variant subdivides the mesh into many more tasks than processors. It uses a hierarchical decomposition to split the data first over nodes, and then over processors within a node. We used different *node* geometries on Thunder's 4-way nodes than on DataStar's 8-way nodes: $4 \times 4 \times 4$ and $4 \times 2 \times 4$, respectively. The *processor* geometries were the same on both platforms: $4 \times 4 \times 4$. Each task relaxes one block and depends on up to seven others from the previous iteration: the block in the current position plus up to six nearest neighbors. In the BASE and OLAP variants each processor stores its mesh as one contiguous memory area. The Thyme variant stores its data as separate contiguous blocks, improving cache locality.

We ran Jacobi3D for 25 iterations on 256 processors using problem sizes varying from 320^3 to 1600^3, beyond which communication is not a significant bottleneck. Fig. 1 (top) shows that the THYME variant enjoys a clear performance advantage over BASE, overlapping 43-78% of the communication on Thunder and 10-80% on DataStar. The OLAP variant not only failed to improve the running time but actually increased it. As noted by Baden and Shalit, the thin outer annulus has large strides and this slows down the updates to cells in the outer annulus considerably [11].

Thyme's graph-based execution model is well suited to this application. Rather than using thin annular faces, we break up the mesh into numerous small cubes, each stored in a contiguous area of memory. Relaxation exhibits good cache locality over these cubes, avoiding the computation time penalty imposed by OLAP's thin outer annulus.

We observed that execution from one iteration was frequently intermingled with that of the next, such that iteration boundaries no longer serve as precise synchronization point. We found no discernible pattern in task execution order from run to run. This implies that flexibility in scheduling may be helping performance and that an optimal schedule may be difficult to predict.

Matrix Multiplication. *MMULT* computes the matrix product $C = A \times B$, formulating the algorithm as a sequence of blocked outer products and subdividing the matrices over a square 2-d processing geometry. The strategy is

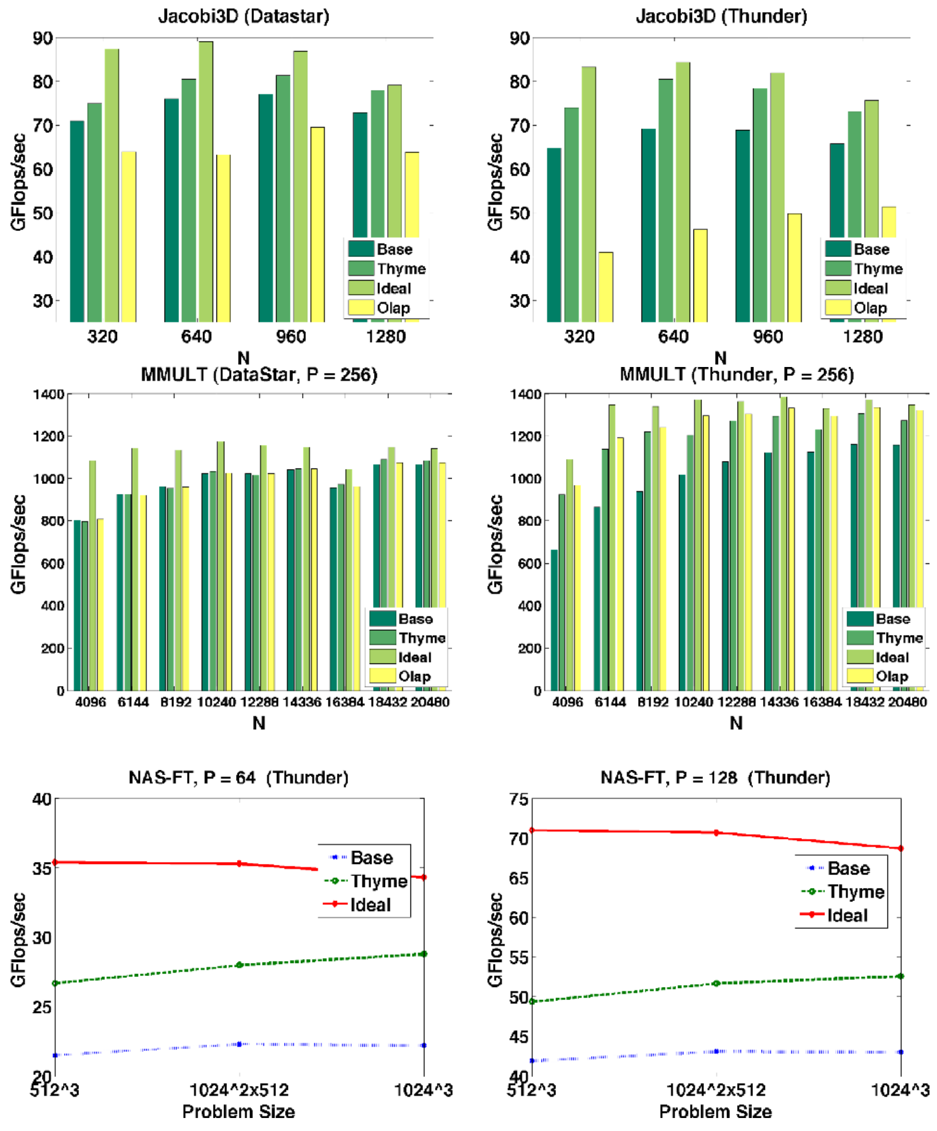


Fig. 1. Data-driven execution improves performance by overlapping communication with computation. Results are shown for Jacobi3D (top), MMULT (middle) running on DataStar (left) and Thunder (right), and for NAS-FT running on 64 and 128 processors of Thunder (bottom, left and right, respectively). No OLAP variant was implemented for FT.

similar to SUMMA [12]. Each submatrix is further subdivided into panels of width (or height) n_b. The algorithm proceeds in $N/(n_b\sqrt{P})$ steps, where the processing geometry has P virtual processors. The panels circulate by row (for A) and by column (for B) and each processor computes a partial matrix product which is summed into its local portion of C (matrix multiply-add, *mpy-add*). Communication involves four nearest neighbors, proceeds in just one direction, and is periodic. BASE performs each mpy-add before transferring a panel of A and B; OLAP initiates panel transfer and then performs the mpy-add in parallel. THYME mirrors the behavior of the OLAP variant; the graph dependencies cause the panels to be passed around their respective rows and columns in pipelined fashion. Each task performs a mpy-add and depends on the left neighbor to receive a panel of A and the upper neighbor to receive a panel of B, passing, after execution, A to the right and B downwards. We used the vendor-provided `dgemm` routine to carry out mpy-adds and used two panels ($n_b = 2$) per block, which was optimal.

We ran MMULT on 256 processors with problem sizes ranging from $4,096^2$ to $20,480^2$, the largest size where communication had a significant affect on performance. Fig. 1 (middle) shows the result. THYME achieves near-optimal performance on Thunder, overlapping 61-93% of the communication. The results are not as clear-cut on DataStar, where the Thyme and baseline variants realize more or less the same performance. This is true because mpy-adds are 17-39% slower than on DataStar than on Thunder (except for the smallest problem size, where DataStar's mpy-adds were faster). Thus, DataStar runs incur a smaller fraction of communication time, and with less communication, there is less benefit to hiding it.

Thyme could not improve on the hand-coded OLAP variant. This is true because the communication pattern is highly constrained to nearest neighbors, and thus the flexibility of Thyme's data driven model doesn't offer an improvement. However, this flexibility doesn't *penalize* performance either. Unlike the hand-coded OLAP variant, the Thyme variant is free from split phase coding and embedded policy decisions, enhancing performance portability.

We used performance meta-data to guide task scheduling in the THYME variant of MMULT. The scheduler's priority queue exhibits LIFO behavior when the task priorities are equal. Thus, newly-ready tasks get scheduled before older readied tasks. While generally beneficial for cache locality, this behavior disrupts the pipeline of block transfers directed by the TaskGraph, tending to serialize computation. Processors would execute tasks over newly arrived blocks, starving neighboring processing nodes and causing long wait times. We were able to improve performance, without having to reformulate the application, by simply assigning older tasks a higher priority than newly ready ones. This entailed annotating the graph with appropriate meta-data. The effect is to alter the scheduler's behavior toward a FIFO without affecting correctness. This behavior was friendlier to the pipeline structure of the graph and resulted in a 20% performance improvement.

NAS-FT. The NAS-FT benchmark includes a costly transpose operation for setting up the FFT, and thus can benefit greatly from masking communication delays. The publicly available code [9] serves as the BASE variant. NAS-FT employs a 1-d virtual processor geometry in the Z dimension such that each processor receives a *slab* of the mesh. Each slab is further subdivided into 2D *sheets* of size $NX \times NY \times 1$, where NX and NY are the leading dimensions. The algorithm has 3 steps. After performing a 2D FFT on each of the NZ sheets, BASE invokes a total exchange `AllToAll` to transpose the data so each processor has slabs of the complete Z dimension. The processors complete the transform by computing 1-d FFTs along the Z dimension. The Thyme variant treats each sheet as a task. Thus, communication is much finer grained than with BASE and is interleaved with communication. The graph dependencies force the final 1-d FFTs to wait for all the 2D transforms to complete, so there are NX 1D transform tasks over sheets of size $1 \times NY \times NZ$.

We measured the execution time for three different problem sizes over various numbers of processors and ran for 20 iterations. The problem sizes were 512^3 (NAS-FT Class C), plus two larger sizes to test the effects of increasing the sheet size ($1024 \times 1024 \times 512$) and the number of sheets (1024^3). Current results are from Thunder only. We did not attempt to reformulate the 2224 line BASE variant to overlap communication with computation (OLAP).

Fig. 1 (bottom) shows that Thyme is able to hide communication significantly–37-65%–depending on the problem size and number of processors. The application incurs a 35-43% communication overhead.

There is room for improvement in NAS FT, in particular, to employ a 2-d "stick" decomposition in lieu of the 1-d decomposition used currently. Thyme is currently implemented using MPI, however, and the difficulty in realizing overlap with a stick decomposition under MPI has been documented [13]. We are investigating an alternative implementation to support sticks and thus improve scalability.

4 Discussion

Graph-based execution models began appearing in the 1970s with classic dataflow [2,3,4]. A large grain variant followed [5]. SMARTS [14] integrated task and data parallelism and provided an API for coarse-grain macro-dataflow. It has been demonstrated on shared memory only. OSCAR[15] had similar goals to SMARTS, but operated on static (compile time) graphs. CILK[16] and Mentat[17] treated functional parallelism. SciRun[18] and UIntah[19] support graphical composition of data flow graphs of components and dynamic load balancing of task graphs. Tarragon (Cicotti and Baden) [20] supports fine grain communication and was originally targeted to cell microphysiology. Husbands and Yelick [21] demonstrated thread scheduling techniques for tolerating latency in dense LU factorization.

In the context of this prior work (and similar to Tarragon) Thyme's contribution is a systematic approach to supporting latency tolerance on distributed memory via the dataflow model, and the ability to modify scheduling behavior by annotating the graph with meta-data. Compared with traditional split phased encoding, Thyme provides an abstract description of the underlying virtual process structure that may be manipulated to optimize execution

To understand the role that the graph can play in tolerating latency, consider Charm++ [22] and Adaptive MPI [23]. Charm++ supports overlap via processor virtualization and asynchronous remote method invocation on shared C++ objects. AMPI (Adaptive MPI) employs virtualized MPI processes and is built on top of Charm++. None of these models employ an explicit dataflow graph to realize overlap. Charm++ employs a more general model than Thyme, embedding dependence information in the form of remote method invocations involving global objects. The task structure is implicit, however, and cannot be manipulated as a free-standing object as with Thyme. The Charm++ developers allude to difficulties with an implicit call graph, in particular, in coupling multiple graphs. Charisma [24] was developed to meet this need.

When a Charm++ method invocation blocks, or an AMPI receive blocks, the thread yields to another, which may block for the same reason. Indeed, our efforts to run an MPI variant of Jacobi3D under AMPI on DataStar failed to improve performance–and actually slowed it down in some cases. Thyme's metadata offer an improvement over virtualization by taking the guesswork out of scheduling. They inform the scheduler about tasks that have all their input data ready, avoiding the guesswork of the "block and yield" model. If the number of cores per processor continues to grow over time, then the significance of informed scheduling will continue to grow as well.

5 Conclusions and Future Work

We have demonstrated that a data-driven formulation enables an application to tolerate latency without embedding scheduling and other policy decisions into the code. The approach provides an opportunity for increased performance because it allows the user to experiment with alternative implementation policies, including application-specific task scheduling in MMULT and tuned data decompositions in Jacobi3D. Thyme admits the use of newly arrived data to enable computation, masking the latency of data still in transit.

Our current implementation is restricted to Cartesian geometries, but covers a range of uniform and irregular problems including: uniform and multilevel finite-difference methods, such as structured adaptive mesh refinement and multigrid, and mesh-based particle methods (but not "tree codes" [25]). So called unstructured finite element methods need a different type of iteration space representation, but the general principles apply.

Systems with many core CPUs will benefit from Thyme's programming model, in particular, thread-aware schedulers that treat symbiosis and cache locality. A steady growth in on-chip parallelism will put pressure on communication subsystems, while at the same time providing an opportunity to optimize execution by expending inexpensive processing cycles.

Although the Thyme API is compact–only a couple of thousand lines of C++–programmers can obtain the benefits of the model without having to learn an entirely new API. We envision that Thyme will become part of a run time support library for a compiler or application library. To this end, we are currently investigating source-to-source translation techniques using Quinlan's ROSE [26] infrastructure. ROSE enables the user to access and transform the abstract syntax tree (AST); full knowledge of function arguments and dependence information is available for subsequent analysis and transformation. Translation support can realize semantic level optimizations on the Thyme library classes and automatically generate calls to the Thyme API. The application programmer can thereby obtain the benefits of Thyme's graph-driven execution model while remaining aloof of many of the details.

Acknowledgments. The authors wish to thank Daniel Quinlan for supervising Jacob Sorensen during visits to the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. Dan's contributions to this work were invaluable. This work was supported by the United States Department of Energy under Award Number DE-FG02-05ER25706 and Prime Contract No. W-7405-ENG-48. Computer time on DataStar was provided by the San Diego Supercomputer Center and the University of California, San Diego.

References

1. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, Argonne, IL (1997), `http://www.mcs.anl.gov/mpi/mpich/`
2. Dennis, J.: Data flow supercomputers. IEEE Computer 13(11), 48–56 (1980)
3. Gurd, J.R., Kirkham, C.C., Watson, I.: The Manchester prototype dataflow computer. Communications of the ACM 28(1), 34–52 (1985)
4. Arvind: Executing a program on the mit tagged-token dataflow architecture. IEEE Transactions on Computers 39(3), 300–318 (1990)
5. Babb II, R.G.: Parallel processing with large-grain data flow technique. Computer 17(7), 55–61 (1984)
6. Kelly, P., Beckmann, O., Field, A., Baden, S.: Themis: Component dependence metadata in adaptive parallel applications. Parallel Processing Letters 11(4), 455–470 (2001)
7. Little, J.D.C.: A proof of the queuing formula L = λ W. Operations Research 9, 383–387 (1961)
8. Colella, P.: Defining software requirements for scientific computing (2004)
9. NASA Advanced Supercomputing Division: NAS parallel benchmarks, `http://www.nas.nasa.gov/Resources/Software/npb.html`
10. Sawdey, A.C., O'Keefe, M.T., Jones, W.B.: A general programming model for developing scalable ocean circulation applications. In: Proceedings of the ECMWF Workshop on the Use of Parallel Processors in Meteorology (January 1997)
11. Baden, S.B., Shalit, D.: Performance tradeoffs in multi-tier formulation of a finite difference method. In: Proc. 2001 International Conference on Computational Science, San Francisco, CA (May 2001)

12. van de Geign, R., Watts, J.: SUMMA: Scalable universal matrix multiplication algorithm. Concurrency: Practice and Experience 9(4), 255–274 (1997)
13. Iancu, C.C., Strohmaier, E.: Optimizing communication overlap for high-speed networks. In: Proc. 12th ACM SIGPLAN Symp on Principles and Practice of Parallel Prog (PPoPP 2007), pp. 35–45. ACM Press, New York (2007)
14. Vajracharya, S., Karmesin, S., Beckman, P., Crotinger, J., Malony, A., Shende, S., Oldehoeft, R., Smith, S.: Smarts: Exploting temporal locality and parallelism through vertical execution. In: International Conference on Supercomputing (1999)
15. Kasahara, H., Yoshida, A.: A data-localization compilation scheme using partial-static task assignment for fortran coarse-grain parallel processing. Parallel Computing 24, 579–596 (1998)
16. Leiserson, C., Randall, K., Zhou, Y.: Cilk: An efficient multithreaded runtime system. In: Proc. Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), Santa Barbara, CA (July 1995)
17. Grimshaw, A.S., Weissman, J.B., Strayer, W.T.: Portable run-time support for dynamic object-oriented parallel processing. ACM Transactions on Computer Systems 14(2), 139–170 (1996)
18. Johnson, C., Parker, S., Weinstein, D., Heffernan, S.: Component-based, problem solving environments for large-scale scientific computing. Concurrency and Computation: Practice and Experience 14(13-15), 1337–1349 (2002)
19. McCorquodale, J., de St. Germain, D., Parker, S., Johnson, C.: The uintah parallelism infrastructure: A performance evaluation. In: High Performance Computing 2001, Seattle (March 2001)
20. Cicotti, P., Baden, S.B.: Asynchronous programming with tarragon. In: Proc. 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France (June 2006)
21. Husbands, P., Yelick, K.: Multithreading and one-sided communication in parallel lu factorization. In: SC 2007: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, Reno, Nevada. ACM, New York (2007)
22. Phillips, J.C., Zheng, G., Kumar, S., Kalé, L.V.V.: NAMD: Biomolecular simulation on thousands of processors. In: Proceedings of SC 2002 (2002)
23. Huang, C., Lawlor, O., Kalé, L.: Adaptive mpi. In: Rauchwerger, L. (ed.) LCPC 2003. LNCS, vol. 2958. Springer, Heidelberg (2004)
24. Huang, C., Kale, L.: Charisma: orchestrating migratable parallel objects. In: HPDC 2007: Proceedings of the 16th international symposium on High performance distributed computing, pp. 75–84. ACM Press, New York (2007)
25. Warren, M.S., Salmon, J.K.: A parallel hashed oct-tree n-body algorithm. In: Supercomputing, pp. 12–21 (1993)
26. Quinlan, D., Miller, B., Philip, B., Schordan, M.: A c++ infrastructure for automatic introduction and translation of openmp directives. In: Proc. 16th Inte. Parallel and Distrib. Proc. Symp., pp. 105–114 (April 2002)

New Optimal Load Allocation for Scheduling Divisible Data Grid Applications

M. Othman*, M. Abdullah, H. Ibrahim, and S. Subramaniam

Department of Communication Technology and Network,
University Putra Malaysia, 43400 UPM Serdang, Selangor D.E., Malaysia
mothman@fsktm.upm.edu.my, monabdullah@hotmail.com

Abstract. In many data grid applications, data can be decomposed into multiple independent sub-datasets and distributed for parallel execution and analysis. This property has been successfully employed by using Divisible Load Theory (DLT), which has been proved as a powerful tool for modeling divisible load problems in data-intensive grid. There are some scheduling models have been studied but no optimal solution has been reached due to the heterogeneity of the grids. This paper proposes a new model called Iterative DLT (IDLT) for scheduling divisible data grid applications. Recursive numerical closed form solutions are derived to find the optimal workload assigned to the processing nodes. Experimental results show that the proposed IDLT model obtains better solution than other models (almost optimal) in terms of *makespan*.

Keywords: Divisible Load Theory, Data Grid, Load Balancing.

1 Introduction

In the last decade, data grids have increasingly become popular for a wide range of scientific and commercial applications [1]. Load balancing and scheduling play a critical role in achieving high utilization of resources in such environments [2]. Scheduling an application is significantly complicated and challenging because of the heterogeneous nature of a grid system. Grid scheduling is defined as the process of making scheduling decisions involving allocating jobs to resources over multiple administrative domains [8,9]. Most of the scheduling strategies try to reduce the *makespan* or the maximum completion time of the task which is defined as the difference between the time when the job was submitted to a computational resource and the time it completed. *makespan* also includes the time taken to transfer the data to the point of computation if that is allowed by the scheduling strategy [5].

In other hand, in many data intensive grid applications, data can be decomposed into multiple independent sub datasets and distributed for parallel

* The author is also an associate researcher at the Lab of Computational Science and Informatics, Institute of Mathematical Research (INSPEM), University Putra Malaysia.

execution and analysis. High Energy Physics (HEP) experiments fall into this category [7]. HEP data are characterized by independent events, and therefore this characteristic can be exploited when parallelizing the analysis of data across multiple sites. In [11], the DLT paradigm has emerged as a powerful tool for modelling data-intensive computational problems incorporating communication and computations issues [5]. An example of this direction is the work by [3] where the DLT is applied to model the grid scheduling problem involving multiple sources to multiple sinks. In that model, they did not consider the communication time. Whereas, the scheduling in grid applications must consider communication and computation simultaneously to achieve high performance. Some related materials to the problem addressed in this paper can be found in [4,7,8,9,11].

2 Scheduling Model

We consider the problem of scheduling large-volume loads (divisible loads) within in multiple sites. Communication is assumed to be predominant between such cluster nodes and is assumed to be negligible within a cluster node [3,4]. This section describes the scheduling model, the notations, the cost model, and the optimality criterion that are used in our research.

We use the scheduling model that was used by [3,4,8]. It can be described as follows. The target data intensive applications model can be decomposed into multiple independent sub tasks and executed in parallel across multiple sites without any interaction among sub tasks [5]. Lets consider job decomposition by decomposing input data objects into multiple smaller data objects of arbitrary size and processing them on multiple virtual sites. For example in theory, the HEP jobs are arbitrarily divisible at event granularity and intermediate data product processing granularity. Assume that a job requires a very large logical input data set D consists of N physical datasets and each physical dataset (of size L_k) resides at a data source (DS_k, for all $k = 1, 2, \ldots, N$) of a particular site. Figure 1 shows how D is decomposed onto networks and their computing resources.

The scheduling problem is to decompose D into datasets (D_i for all $i = 1, 2, \ldots, M$) across M virtual sites in a Virtual Organization given its initial physical decomposition. We assume that the divisible data can be analyzed at any site.

2.1 Notations and Definitions

All notations and their definitions used throughout this paper are shown in Table 1.

2.2 Cost Model

The execution time cost (T_i) of a subtask allocated to the site i and the turn around time ($T_{Turn\_Around\_Time}$) of a job J can be expressed as follows

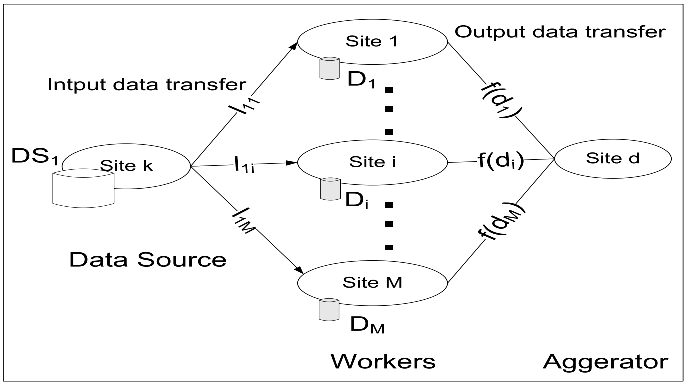


Fig. 1. Data decomposition and their processing

Table 1. Notation and Definition

Notation	Definition
M	The total number of nodes in the system
L	The loads in data file
α_i	The fraction of load that node i will receive from the data file
L_i	The amount of load that node i will receive from the data file
w_j	The inverse of the computing speed of node i
Z_in_i	The link between node i and the data source
Z_out_i	The link between node i and the aggrator
T_i	The processing time in node i

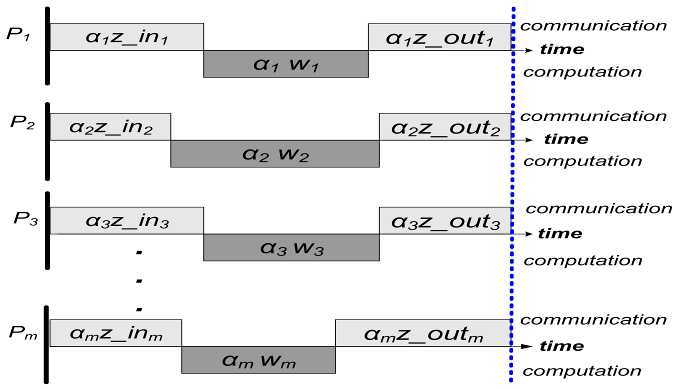


Fig. 2. The communication and computation of sources within the system (optimal case)

$$T_i = T_{input\_cm}(i) + T_{cp}(i) + T_{output\_cm}(i,d)$$

and

$$T_{Turn\_Around\_Time} = \max_{i=1}^{N}\{T_i\},$$

respectively. The input data transfer $T_{input\_cm}(i)$, computation $T_{cp}(i)$, and output data transfer to the client at destination site d, $T_{output\_cm}(i,d)$ are presented as

$$T_{input\_cm}(i) = L_i \cdot \frac{1}{Z_i}, \quad T_{cp}(i) = L_i \cdot w_i \cdot ccRatio$$

and

$$T_{output\_cm}(i,d) = f(L_i) \cdot Z_{id}$$

respectively. Where $L_i = L \cdot \alpha_i$ and the function $f(d_i)$ is an output data size and *ccRatio* is the non-zero ratio of computation and communication. The turn around time of an application is the maximum among all the execution times of the sub tasks.

The problem of scheduling a divisible job onto M sites can be stated as deciding the portion of original workload (D) to be allocated to each site, that is, finding a distribution of α_i which minimizes the turn around time of a job. The proposed model uses this cost model when evaluating solutions at each generation.

2.3 Optimality Criterion

In all literatures related to divisible load scheduling [7,9], an optimality criterion is used to derive an optimal solution which stated as follows. It states that in order to obtain an optimal processing time, it is necessary and sufficient that all the sites that participate in the computation must stop at the same time. Otherwise, load could be redistributed to improve the processing time. The timing diagram for this distributed system in optimal case is depicted in Fig. 2.

3 Proposed IDLT Model

The load scheduling problem is to decompose D into datasets (D_i for all $i = 1, 2, \ldots, M$) across M virtual sites in a Virtual Organization given its initial physical decomposition. This model includes two steps

3.1 Initial Solution

The proposed model will start from a good initial solution. ADLT and A^{2}DLT models will be used for this purpose. The best solution (minimum *makespan*) will be considered as an initial solution of the iterative model. As it is explained in [8,9,11], ADLT and A^{2}DLT models produced good results for computation and communication intensive applications, respectively.

3.2 The Iterative Model

The optimality criterion that discussed in section 2.3 will be used in the design of the load distribution strategy. The IDLT model involves the following steps

1. First, we divide the load using one of the adaptive DLT models.
2. Calculate the *makespan* using the cost model.
3. If all nodes finish at the same time, go to step 8 else go to step 4.
4. Then, we calculate the summation (Sum) of the processing time of the nodes.
5. Next, we calculate the average time by $avg = Sum/M$.
6. After that, we redistribute the load depending on the average (avg) based on the iterative numerical equation that will be discussed later in this section.
7. Go to step 2
8. The current time is the final *makespan*.
9. End

Here, we will discuss step by step the derivation of a closed form equation by which one can calculate the optimal fraction of the load that has to be assigned to each processing node in order to achieve the minimum *makespan* and the optimal data allocation for each processor. The processing nodes (w_i), communication links (Z_i)and applications types ($ccRatio$) were assumed to have different values.

After we calculate the *makespan* as an initial solution to the IDLT model, we will compute the $\sum_{i=1}^{M} T_i$ where M is the number of the processing nodes. Based on cost model, we have

$$T_i = L_i \cdot z\_in_i + w_i \cdot L_i \cdot ccratio + L_i \cdot z\_out_i \tag{1}$$

where z_in_i and z_out_i is the input communication time and output communication time of link i, respectively. Then, compute the average of completion time as

$$avg = \frac{\sum_{i=1}^{M} T_i}{M}. \tag{2}$$

In any iteration, the *makespan* of any nodes must equal to (avg) in order to get the optimal solution. It means that, the load is distributed among the processing node M equally. While the average time is the processing time of load α that is calculated by (1), we can recalculate the amount of α if we have the (avg). In general, if avg is

$$avg = \alpha \cdot z + \alpha \cdot w \tag{3}$$

The load fraction α can be calculated by (4),

$$\alpha = \frac{avg}{z + w} \tag{4}$$

In this case, for any iteration, after we calculate the time average of processing any load, we can calculate the amount of this load. Furthermore, if the processing time of the first node to process α_1 is

$$avg = \alpha_1 \cdot z\_in_1 + \alpha_1 \cdot w_1 \cdot ccRatio + \alpha_1 \cdot z\_out_1 \tag{5}$$

The α_1 will be calculated by:

$$\alpha_1 = \frac{avg}{z\_in_1 + (w_1 \cdot ccRatio) + z\_out_1} \tag{6}$$

Consequently, the α_2 of the second node will be calculated as

$$\alpha_2 = \frac{avg}{z\_in_2 + (w_2 \cdot ccRatio) + z\_out_2} \tag{7}$$

Finally, the α_M of the Mth node will be

$$\alpha_M = \frac{avg}{z\_in_M + (w_M \cdot ccRatio) + z\_out_M} \tag{8}$$

Equation (8) is correct only in the last iteration (when we get the optimal solution). But in the first iteration, the last node will take the rest of the load only as

$$\alpha_M = (1 - (\alpha_1 + \alpha_2 + \cdots + \alpha_{M-1})) \cdot L \tag{9}$$

There are M-1 equations. An additional equation is called a normalization equation, which states that the summation of the all the allocation fractions should be 1.

$$\alpha_1 + \alpha_2 + \cdots + \alpha_M = 1 \tag{10}$$

The load of the last node does not equal the load that produces the (avg) time. Thats means, the last node still does not take the optimal load. Here, we will compute the new *makespan* (the *makespan* will be calculated by cost model that discussed in Section 2.2 every iteration).

Consequently, we will carry out these steps in the second iteration. In this iteration, the *makespan* will be reduced but still not optimal. Therefore, we will carry out these steps until certain termination condition happens. The termination condition here is the optimality criterion. All nodes must finish the load processing at the same time.

The IDLT model for single source produces almost optimal solution after some iterations. There is a very small different among the nodes processing time (small fractions). In this model, all nodes will take the new load based on the new average. It means that all nodes will finish at the same time. For the last node will take the rest of load without considering the average. For the next iteration, the new average will be reduced. The last node will take more load than previous iteration. After some iterations the last node will finish as same time as the others.

4 Experimental Results

To measure the performance of the proposed IDLT model against CDLT, ADLT and A^{2}DLT models, randomly generated experimental configurations were used [5,8,9]. The estimated expected execution time for processing a unit dataset on

Table 2. Results IDLT Model: Step-by-Step

Steps	Node 1	Node 2	Node 3	Node 4	Node 5
0	5643.84	1440.41	2993.02	843.48	1108.35
5	1822.61	1822.61	1822.61	1822.61	1757.71
10	1799.71	1799.71	1799.71	1799.71	1798.61
11	1799.49	1799.49	1799.49	1799.49	1799.00

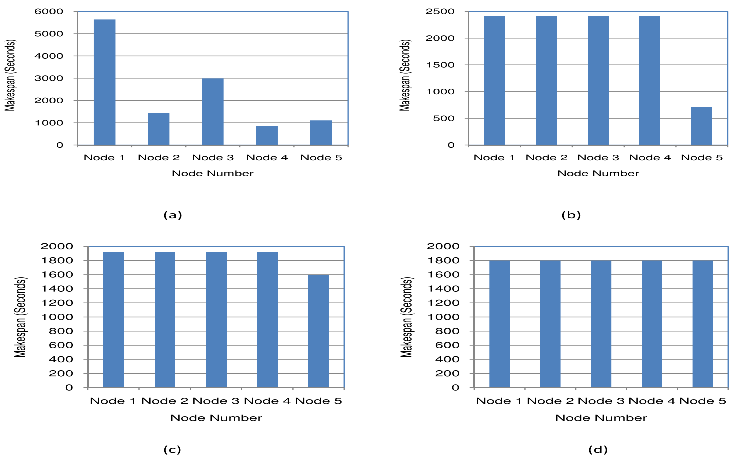


Fig. 3. Comparison of various iterations for IDLT model, (a) initial step, (b) and (c) intermediate steps, and (d) final step

each site, the network bandwidth between sites, input data size, and the ratio of output data size to input data size were randomly generated with uniform probability over some predefined values. The network bandwidth between sites is uniformly distributed between 1 Mbps and 10 Mbps.

We examined the overall performance of each model by running them under 100 randomly generated Grid configurations. We varied the parameters, *ccRatio* (0.001 to 1000), r_{cb} (10 to 500), and M (1 to 100). Thus, from series of experiments, it can be concluded that the IDLT gives an optimal solution and all nodes stop at the same time.

Let considers the system and loads with processing nodes M=5 and *ccRatio* =0.001, respectively. Using the IDLT model, we have the experimental results as shown in Table 2 and Fig. 3. In an initial solution, the maximum completion time or *makespan* is 5643.84 seconds. It is reduced until *makespan* 1799 seconds. In the last step, all nodes finished the processing at the same time (at 1799 Seconds). Furthermore, when we increase the number of iteration, all nodes will finish at almost the same time (1799.32 seconds) and it is improved approximately 68%.

Figure 3 showed that, the five processing nodes finish at almost same time. It means that, in the last iteration, the load is distributed to the processing node almost equally.

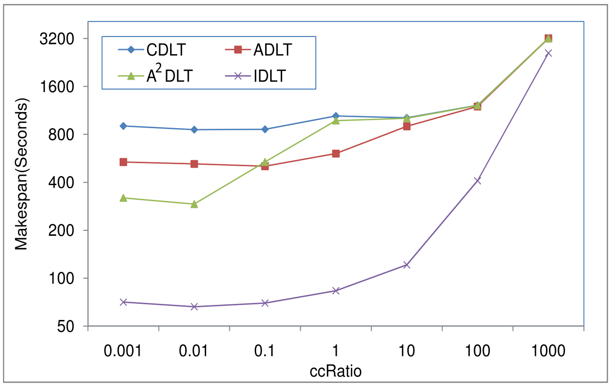


Fig. 4. *Makespan* vs. *ccRatio* for CDLT, ADLT, A^{2}DLT and IDLT(M=100)

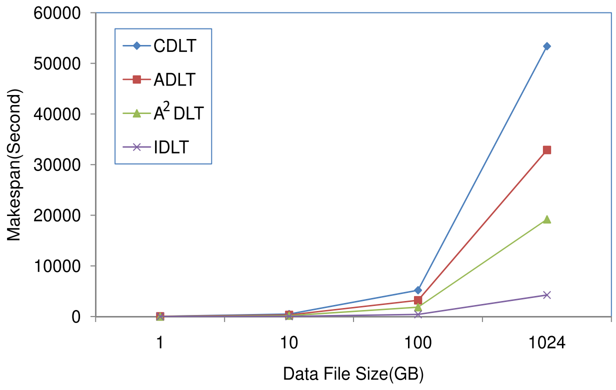


Fig. 5. *Makespan* vs. data file size for CDLT, ADLT, A^{2}DLT and IDLT(M=100 and *ccRatio*=0.001)

To show how these models perform on different type of application (different *ccRatio*=100), we executed the model and plotted all the results as shown in Fig. 4. From the plotted graph, the IDLT model is the best for any type of application. As expected, the IDLT model produce the almost optimal solution from single source.

Different size of data files are used considering large scale data grid. It is varied from 1 GB to 1 TB. Figure 5 clearly showed that the IDLT model produce better results for all sizes.

The convergence metric records how the initial *makespan* value minimized during the iteration between the initial and optimal solutions. Figure 6 depicts the convergence of the models for the average out of ten executions for when the *ccRatio* is equal to 1000. All models have same results, whereas the result of IDLT model is significantly reduced as the number of iteration increased.

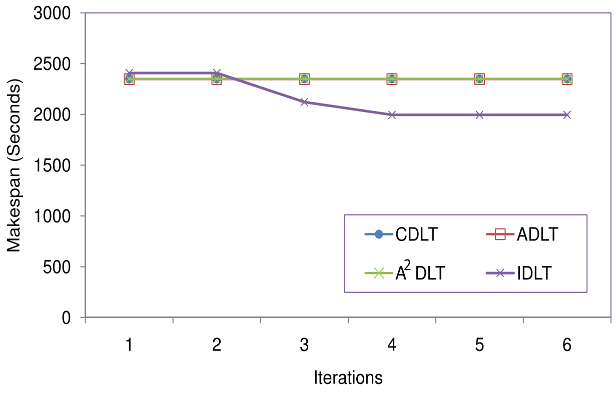


Fig. 6. Convergence for IDLT model for single source (*ccRatio*=1000))

5 Conclusion

In this paper, we have developed an effective iterative model for optimal divisible load allocation. The IDLT model is proposed for load allocation to processors and links for scheduling divisible data grid applications. The experimental results showed that the proposed IDLT model is capable of producing almost optimal solution for single source scheduling. Hence, the proposed model can balance the processing loads efficiently and can be embedded into the existing data grid schedulers.

References

1. Tierney, B., Johnston, W., Lee, J., Thompson, M.: A Data Intensive Distributed Computing Architecture for Grid Applications. Future Generation Computer Systems 16(5), 473–481 (2000)
2. Xiao, Q.: Design and Analysis of a Load Balancing Strategy in Data Grids. Future Generation Computer Systems 16(23), 132–137 (2007)
3. Tang, M., Lee, B.-S., Tang, X., Yeo, C.-K.: The Impact of Data Replication on Job Scheduling Performance in the Data Grid. Future Generation Computer Systems 22(3), 254–268 (2006)
4. Wong, H.M., Veeravalli, B., Dantong, Y., Robertazzi, T.G.: Data Intensive Grid Scheduling: Multiple Sources with Capacity Constraints. In: Proceeding of the IASTED Conference on Parallel and Distributed Computing and Systems, Marina del Rey USA, 7-11 (2003)
5. Kim, S., Weissman, J.B.: A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications. In: IEEE Proceeding of the International Conference on Parallel Processing, Washington DC, USA, vol. 1, pp. 406–413 (2004)
6. Venugopal, S., Buyya, R., Ramamohanarao, K.: A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing. ACM Computing Surveys 38(1), 1–53 (2006)
7. Abraham, A., Buyya, R., Nath, B.: Nature's Heuristics for Scheduling Jobs on Computational Grids. In: Proceedings of 8th IEEE International Conference on Advanced Computing and Communications, pp. 45–52 (2000)

8. Othman, M., Abdullah, M., Ibrahim, H., Subramaniam, S.: Adaptive Divisible Load Model for Scheduling Data-Intensive Grid Applications. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 446–453. Springer, Heidelberg (2007)
9. Othman, M., Abdullah, M., Ibrahim, H., Subramaniam, S.: A^2DLT: Divisible Load Balancing Model for Scheduling Communication-Intensive Grid Applications. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 246–253. Springer, Heidelberg (2008)
10. Viswanathan, S., Veeravalli, B., Robertazzi, T.G.: Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems. IEEE Transaction of Parallel and Distributed Systems 18(10), 1450–1461 (2007)
11. Bharadwaj, V., Ghose, D., Robertazzi, T.G.: Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems. Cluster Computing 6(1), 7–17 (2003)

Dynamic Resizing of Parallel Scientific Simulations: A Case Study Using LAMMPS

Rajesh Sudarsan[1], Calvin J. Ribbens[1], and Diana Farkas[2]

[1] Department of Computer Science, Virginia Tech, Blacksburg, VA 24060
sudarsar@vt.edu, ribbens@vt.edu
[2] Department of Materials Science and Engineering, Virginia Tech, Blacksburg, VA 24061

Abstract. Large-scale computational science simulations are a dominant component of the workload on modern supercomputers. Efficient use of high-end resources for these large computations is of considerable scientific and economic importance. However, conventional job schedulers limit flexibility in that they are 'static', i.e., the number of processors allocated to an application can not be changed at runtime. In earlier work, we described ReSHAPE a system that eliminates this drawback by supporting dynamic resizability in distributed-memory parallel applications. The goal of this paper is to present a case study highlighting the steps involved in adapting a production scientific simulation code to take advantage of ReSHAPE. LAMMPS, a widely used molecular dynamics code, is the test case. Minor extensions to LAMMPS allow it to be resized using ReSHAPE, and experimental results show that resizing significantly improves overall system utilization as well as performance of an individual LAMMPS job.

Keywords: LAMMPS, parallel clusters, dynamic scheduling, data redistribution.

1 Introduction

Today's terascale and petascale computers exist primarily to enable large-scale computational science and engineering simulations. While high-throughput parallel applications are important and often yield valuable scientific insight, the primary motivation for high-end supercomputers is applications requiring hundreds of compute cores, with data sets distributed across a large aggregate memory, and with relatively high interprocess communication requirements. Such calculations are also characterized by long running times, often measured in weeks or months. In this high-capability computing context, efficient utilization of resources is of paramount importance. Consequently, considerable attention has been given to issues such as parallel cluster scheduling and load balancing, data redistribution, performance monitoring and debugging, and fault-tolerance. The goal is to get the maximum amount of science and engineering insight out of these powerful (and expensive) computing resources.

A constraint imposed by existing cluster schedulers is that they are 'static,' i.e., once a job is allocated a set of processors, it continues to use those processors until it finishes execution. Even if there are idle processors available, parallel applications cannot use them because the scheduler cannot allocate more processors to an application at runtime. A more flexible approach would allow the set of processors assigned to a job to be expanded or contracted at runtime. This is the focus of our research—dynamically reconfiguring, or *resizing*, parallel applications.

We are developing *ReSHAPE*, a software framework designed to facilitate and exploit dynamic resizing. In [1] we describe the design and initial implementations of ReSHAPE and illustrate its potential for simple applications and synthetic workloads. An obvious potential benefit of resizing is reduced turn-around time for a single application; but we are also investigating benefits such as improved cluster utilization, opportunities for new priority-based scheduling policies, and better mechanisms to meet quality-of-service or advance reservation requirements. Potential benefits for cluster utilization under various scheduling scenarios and policies are considered in [2]. Efficient data redistribution schemes are described in [3].

In this paper we investigate the potential of ReSHAPE for resizing production computational science codes. As a test case we consider LAMMPS [4,5], a widely used molecular dynamics (MD) simulation code. One of us uses this code on a regular basis to study the mechanical behavior of nanocrystalline structures [6]. In a typical case we run LAMMPS on 100-200 processors for hundreds of hours. LAMMPS has three characteristics typical of most production computational science codes: a large and complex code base, large distributed data structures, and support for file-based checkpoint and recovery. The first two characteristics are a challenge for resizing LAMMPS jobs. However, file-based checkpointing offers a simple but effective way to resize LAMMPS using ReSHAPE. We describe the changes required in the LAMMPS source to use it with ReSHAPE. Experimental results show that resizing significantly improves overall system utilization as well as the performance of an individual LAMMPS job.

Recent research has focused on dynamic reconfiguration of applications in a grid environment [7,8]. These frameworks aim at improving the resources assigned to an application by replacement rather than increasing or decreasing the number of resources. Vadhiyar and Dongarra [9] apply a user-level checkpointing technique to reconfigure applications for the Grid. During reconfiguration, the application writes a checkpoint file. After the application has been migrated to the new set of resources, the checkpointed information is read and redistributed across the new processor set. The DRMS framework proposed by Moreira and Naik [10] also uses file-based data redistribution to redistribute data across a reconfigured processor set. Cirne and Berman [11] describe an application-aware job scheduler for reconfiguring moldable applications. The scheduler requires a user to specify legal processor partition sizes ahead of time.

The remainder of the paper is organized as follows. Section 2 introduces ReSHAPE and describes the modifications required to use LAMMPS with ReSHAPE. We summarize experimental results in Section 3. Section 4 summarizes and concludes the paper.

2 ReSHAPE Applied to LAMMPS

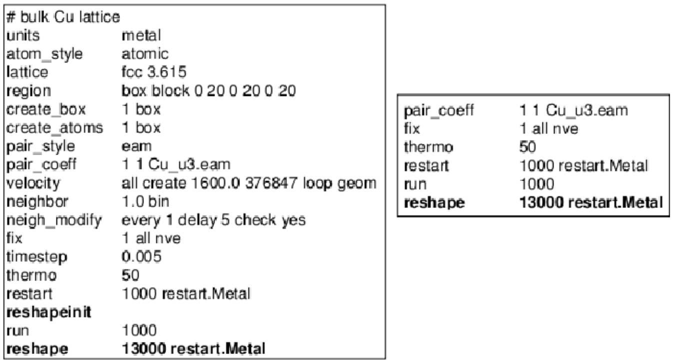
2.1 ReSHAPE Framework

The architecture of the ReSHAPE framework consists of two main components. The first component is an application scheduling and monitoring module which schedules and monitors jobs and gathers performance data in order to make resizing decisions based on scheduling policies, application performance, available system resources, and the state of other running and enqueued jobs. The second component of the framework consists of a programming model for resizing applications. This includes a resizing library and an API for applications to communicate with the scheduler to send performance data and actuate resizing decisions. The resizing library includes algorithms for mapping processor topologies and redistributing data from one processor topology to another. ReSHAPE targets applications that are *homogeneous* in two important ways. First, the approach is best suited to applications where data and computations are relatively uniformly distributed across processors. Second, at a high-level the application should be iterative, with the amount of computation done in each iteration being roughly the same. While these assumptions do not hold for all cluster jobs, they do hold for a significant number of large-scale scientific simulations. A more detailed discussion on ReSHAPE is available in [1,3].

2.2 Extending LAMMPS for Resizing

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical molecular dynamics code which models particles interacting in a liquid, solid, or gaseous state. Like many computational science codes, LAMMPS uses domain-decomposition to partition the simulation domain into 3D sub-domains, one of which is assigned to each processor. LAMMPS is a good candidate for ReSHAPE-enabled resizing since it scales well and includes an outer loop executed many thousands of times, with a relatively constant amount of work done during each iteration of that outer loop. The point at which an iteration of this outer loop finishes is a natural candidate for a *resize* point, i.e., a point at which the code should contact the ReSHAPE scheduler to potentially be resized. To explore the potential of using LAMMPS with ReSHAPE, we modified the LAMMPS source to insert ReSHAPE API calls. By leveraging the existing checkpoint-recovery capabilities of LAMMPS, we can extend the code to support resizing with only a few small changes, described next.

LAMMPS reads an input script to set problem-specific parameter values and runs the simulation. A similar script is used to restart a computation using a restart file. Figure 1 shows a sample ReSHAPE-instrumented LAMMPS input script and restart script, with our changes marked in bold. These input files would be very familiar to a LAMMPS user. Only two additional commands—*reshapeinit* and *reshape*—are needed to support resizing. The LAMMPS command parser must be extended to recognize these two commands. The only other modifications to the existing LAMMPS code base are replacing all occurrences of MPI_COMM_WORLD with RESHAPE_COMM_-WORLD, and including an additional call to the ReSHAPE initialization method in the LAMMPS main program, executed only by newly spawned processes. The *reshapeinit* command in the input script causes ReSHAPE's initialization method to be called. The *reshape* command takes two arguments: the total number of iterations to be run and a restart file name. The existing LAMMPS *restart* command is used to indicate the number of iterations after which a checkpoint file will be generated. We use these restart points as potential resize points for ReSHAPE. In particular, we use the *run* command to specify the number of iterations to execute before stopping. This value must be the same as the value given in the *restart* command, e.g., 1000 in Figure 1. In this way, LAMMPS executes a predetermined number of iterations, generates a restart file, contacts the



```
# bulk Cu lattice
units           metal
atom_style      atomic
lattice         fcc 3.615
region          box block 0 20 0 20 0 20
create_box      1 box
create_atoms    1 box
pair_style      eam
pair_coeff      1 1 Cu_u3.eam
velocity        all create 1600.0 376847 loop geom
neighbor        1.0 bin
neigh_modify    every 1 delay 5 check yes
fix             1 all nve
timestep        0.005
thermo          50
restart         1000 restart.Metal
reshapeinit
run             1000
reshape         13000 restart.Metal
```

```
pair_coeff      1 1 Cu_u3.eam
fix             1 all nve
thermo          50
restart         1000 restart.Metal
run             1000
reshape         13000 restart.Metal
```

Fig. 1. LAMMPS input script (left) and restart script (right) extended for ReSHAPE

ReSHAPE scheduler, and then depending on the response from the scheduler, either does a restart on the current processor set or on some larger or smaller processor set.

The *reshape* command executes a new method which uses the ReSHAPE API to communicate with the ReSHAPE scheduler. Depending on the scheduler's decision, the resizing library expands, contracts or maintains the processor size for an application. The application clears its old simulation box and re-initializes the system with the new processor size. A new method is implemented to rebuild the process universe each time the processor set size changes after resizing. The application re-initializes the system with new parametric values from the restart file and resumes execution. All the newly spawned processes initialize ReSHAPE before receiving the restart information from processor rank 0.

3 Experimental Results and Discussions

This section presents experimental results to demonstrate the potential of dynamic resizing for MD simulation. The experiments were conducted on 50 nodes of Virginia Tech's System X. Each node has two 2.3 GHz PowerPC 970 processors and 4GB of main memory. Message passing was done using OpenMPI [12] over an Infiniband interconnection network.

We present results from two sets of experiments. The first set focuses on benefits of dynamic resizing for individual MD applications to improve their execution turn-around time; the second set looks at the improvements in overall cluster utilization and throughput which result when multiple static and resizable applications are executing concurrently. In our experiments, we use a total of five different applications—LAMMPS plus four applications from the NAS parallel benchmark suite [13]: CG, FT, IS, LU. We use class A and class B problem sizes for each NAS benchmark, for a total of nine different jobs. For LAMMPS we use an EAM metallic solid benchmark problem. The problem computes the potentials among the metallic copper atoms using an embedded atom potential method (EAM) [14]. It uses NVE time integration with a force cutoff of 4.95 Angstroms and has 45 neighbors per atom. The different problem sizes used for the MD benchmark are listed in Figure 2. We generate a workload of 17 jobs from these

Table 1. Job workloads and descriptions

(a) Experiment workloads

Workload	nApps	Applications
W1	17	LMP2048 (1), IS-B (1), FT-A (4), CG-A (3), CG-B (3), IS-A (2), LU-A (1), LU-B (2)
W2	17	LMP256 (2), LMP2048 (1), LMP864 (3), LU-B (1), FT-A (2), FT-B (1), IS-A (1), CG-B (2), CG-A (2), LU-A (1), IS-B (1)

(b) Application description

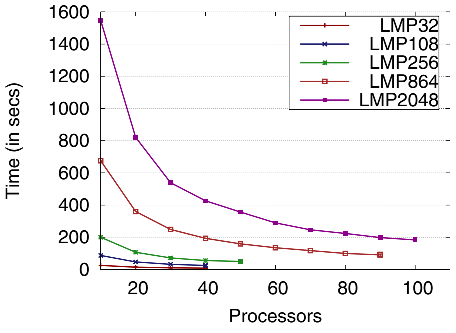
App Name	nProcs	nIters
LMP2048, LMP864	30	16000
LMP256	20	16000
CG-A	16	1000
CG-B	32	40
IS-A, FT-A	8	200
FT-B	32	100
IS-B	16	200
LU-A	64	200
LU-B	32	10

nine job instances, with each newly arriving job randomly selected with equal probability. Table 1(a) lists the different workloads used in our experiments. The number listed in parenthesis for each job is the number of jobs for each application in the job trace for that particular workload. All the LAMMPS jobs execute for 16000 iterations and the timing results are recorded after every 1000 iterations. The jobs reach their resize points after every 1000 iterations. All NAS benchmark applications are configured to expand only in powers-of-2 processor sizes, i.e., 2, 4, 8, 16, 32 and 64. The starting processor size and the number of iterations for each job are listed in Table 1(b). The arrival time for each job in the workload is randomly determined using a uniform distribution between 50 and 650 seconds.

3.1 Performance Benefit for Individual MD Applications

Although in practice it is not always possible to run a single application on an entire cluster, it is not uncommon to have a large number of processors available at some point during a long running application. These processors can be alloted to a running application, so as to probe for a processor configuration beyond which adding more processors will not benefit the application's performance, i.e., to look for a 'sweet-spot' processor allocation for that job. ReSHAPE uses this technique to probe for sweet spots for resizable applications. It uses an application's past performance results, and a simple performance model, to predict whether the application will benefit from additional processors. Based on the prediction, the scheduler decides whether an application should expand, contract or maintain its current processor size.

To get an idea of the potential of sweet spot detection, consider the data in Figure 2, which shows the performance of the LAMMPS benchmark with various problem sizes at different processor configurations. All jobs start with 10 processors and gradually add more processors at every resize point as long as the expand potential [2] of an application remains greater than the fixed threshold performance potential. The threshold value of the performance potential can vary based on the scheduling policy implemented in the system. We observe that the performance benefits due to resizing for small jobs (LMP32 and LMP108) are small; these jobs reach their sweet spots at only 40 processors. As expected, jobs with larger problem sizes show greater performance benefit



Job	Number of atoms
LMP32	32000
LMP108	108000
LMP256	256000
LMP864	864000
LMP2048	2048000

Fig. 2. Identifying the execution sweet spot for LAMMPS. The table shows the number of atoms for different LAMMPS problem sizes.

Table 2. Performance improvement in LAMMPS due to resizing. Time in secs.

Number of Atoms	nResize	Iteration time		Improvement		Overhead	
		Static	ReSHAPE	Time	%	Time	%
32000	3	330.98	168	162.98	49.24	29.31	17.98
108000	3	1132.56	443	689.56	60.72	29.42	4.27
256000	4	2601.30	909	1692.30	65.05	48.25	2.85
864000	8	8778.02	2529	6249.02	71.18	145.68	2.33
2048000	9	20100.86	5498	14602.00	72.64	198.51	1.36

with more processors. For example, LMP864 reaches its sweet spot at 90 processors. LMP2048 shows a performance improvement of 11.6% when expanding from 90 to 100 processors and has the potential to expand beyond 100 processors. The jobs continue to execute at their sweet spot configuration till they finish execution. Table 2 compares the improvement in performance due to resizing for LAMMPS jobs for different problem sizes with the associated overhead. For jobs with smaller problem sizes, the cost of spawning new processors and redistributing data contributes a significant percentage of the total overhead. By reducing the frequency of resize points for smaller jobs, the overhead can be outweighed by performance improvements over multiple iterations at the new processor size. For a production science code such as LAMMPS, ReSHAPE leverages the existing checkpoint-restart mechanism and uses the restart files to redistribute data after resizing. The table shows the number of resize operations performed for each problem size. We observe that as the problem size increases, LAMMPS benefits more from resizing, with relatively low overhead cost. For example, as the problem size increased from 32000 to 2048000 atoms, the performance improvement increased from 49.24% to 72.64% whereas the redistribution overhead decreased from 17.98% to 1.36%. The performance improvements listed in Table 2 include the resizing overhead.

3.2 Performance Benefits for MD Applications in Typical Workloads

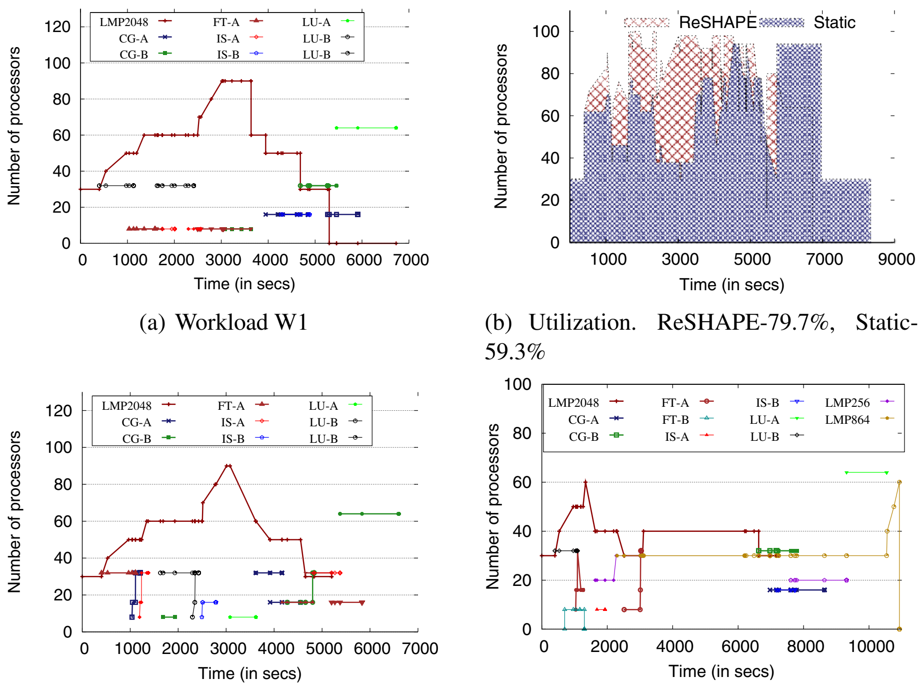
The second set of experiments involves concurrent scheduling of a variety of jobs on a cluster using the ReSHAPE framework. We can safely assume that when multiple jobs are concurrently scheduled in a system and are competing for resources, not all jobs can adaptively discover and run at their sweet spot for the entirety of their execution. In this experiment, we use a *FCFS* + *least-impact* policy to schedule and resize jobs on the cluster. Under this policy, arriving jobs are scheduled on a first-come, first-served basis. If there are not enough processors available to schedule a waiting job, the scheduler tries to contract one or more running jobs to accommodate the queued job. Jobs are selected for contraction if the ReSHAPE performance monitor predicts those jobs will suffer minimal performance degradation by being contracted. More sophisticated policies are available in ReSHAPE and are discussed in detail in [2].

We illustrate the performance benefits for a LAMMPS job using three different scenarios. The first scenario uses workload W1 and schedules a single resizable LAMMPS job with 16 static jobs. The second scenario also uses W1 but schedules all jobs as resizable jobs. The third scenario uses workload W2 to schedule 6 instances of LAMMPS with 11 instances of CG, FT, IS and LU as resizable jobs.

Figure 3(a) shows the processor allocation history for the first scenario. LMP2048 starts execution at t=0 seconds with 30 processors. At each resize point, LMP2048 tries to expand its processor size by 10 processors if there are no queued jobs. The granularity with which a resizable application expands at its resize point is set as a configuration parameter in ReSHAPE. At t=3017 seconds, LMP2048 expands to 90 processors and maintains that size until t=3636 seconds when a new static job, CG-B, arrives with a processor request of 32 processors. The scheduler immediately contracts LMP2048 at its next resize point to 60 processors to accommodate the new job. At t=3946 seconds, LMP2048 further contracts to 50 processors to accommodate a CG-A application. Finally, LMP2048 reduces to its starting processor size to accommodate another CG-B application at t=4686 seconds. Due to the lack of additional processors, LMP2048 maintains its starting processor size till it finishes its execution. Figure 3(b) compares system utilization using ReSHAPE with static scheduling. Static scheduling requires a total of 8837 seconds to complete the execution for all the jobs whereas ReSHAPE finishes the execution in 6728 seconds. An improvement of 20.4% in system utilization due to dynamic resizing translates into a 36.4% improvement in turn-around time for LMP2048.

Figure 3(c) shows the processor allocation history for the second scenario where all the jobs are resizable. Similar to the first scenario, LMP2048 starts execution at t=0 seconds and gradually grows to 90 processors. Due to contention for resources among jobs, all CG, LU, FT-B and IS-B jobs run at their starting processor configuration. IS-A and FT-A are short running jobs and hence they are able to resize quickly and grow up to 32 processors. Although LMP2048 is a long running job, it is able to expand because of its small and flexible processor reconfiguration requirement at its resize point. LMP2048 contracts to 60 processors at t=3618 seconds and further to 50 processors at t=3916 seconds to accommodate two CG applications. It contracts to its starting processor size at t=4655 seconds and maintains the size till it finishes execution. For the same workload W1, ReSHAPE finishes the execution of all the jobs in 6588 seconds with an overall system utilization of 75.3%.The turn-around time for LMP2048 improves by 36.8%.

Figure 3(d) illustrates a third job mix scenario where multiple instances of the LAMMPS application and applications from the NAS benchmark suite are scheduled



(a) Workload W1

(b) Utilization. ReSHAPE-79.7%, Static-59.3%

(c) Workload W1 (All resizable)

(d) Workload W2 (All resizable)

Fig. 3. Processor allocation and overall system utilization for a job mix of static and resizable LAMMPS jobs

Table 3. Performance results of LAMMPS jobs for jobmix experiment. Time in secs.

	LAMMPS jobs	Job completion time			
		ReSHAPE	Static	Improvement	%
Scenario 1	LMP2048	5301	8337	3036	36.4
Scenario 2	LMP2048	5268	8337	3069	36.8
Scenario 3	LMP2048	7239	8469	1257	15.0
	LMP864	3904	3726	-178	-5.0
	LMP864	3892	3763	-129	-3.0
	LMP864	3756	3744	-12	-0.3
	LMP256	1370	1592	222	14.0
	LMP256	1704	1594	-110	-7.0

together as resizable applications. LMP2048 increases from 30 to 60 processors at 1337 seconds and contracts immediately to its starting processor size to allow scheduling of other queued jobs. LMP2048 expands again and maintains its processor size at 40 till it finishes execution. LMP256 expands from its starting processor configuration to 30 processors and maintains its size at 30 till completion. LMP864 starts its execution at t=7173 seconds, and executes at its starting processor size due to unavailability of additional processors. It expands to 40 and 50 processors at t=10556 seconds and t=10760 seconds, respectively, and finishes with 60 processors at t=10929 seconds. The remaining LAMMPS jobs execute at their starting processor configuration. Static scheduling finishes the execution of W3 in 10997 seconds whereas ReSHAPE requires only 10929 seconds. With ReSHAPE, the overall utilization is 88.1%, an improvement of 9.3% compared to static scheduling.

Table 3 summarizes the performance improvements for LAMMPS jobs in all three scenarios. The performance of LMP2048 improved by 36.4% and 36.8% in scenarios 1 and 2, respectively, whereas in scenario 3 the performance improved by only 15%. We observe that LMP864 and LMP256 jobs perform better with static scheduling than with ReSHAPE. The degradation in performance is due to the data redistribution overhead incurred by the application at each resize point. In the third scenario, all three LMP864 and one LMP256 jobs execute either at their starting processor configuration or close to it. In our current implementation, LAMMPS considers each run to the resize point as an independent execution step. It requires a restart file to resume the execution after resizing. The restart file is required even when the application does not resize at its resize point. Thus, an additional overhead cost is incurred in writing and reading the restart files resulting in performance degradation. LMP256 suffers a maximum performance degradation of 7% whereas LMP864 loses performance by 5% compared to static scheduling. However, note that the statically scheduled LAMMPS jobs did not write any restart files, which in practice is unusual.

4 Conclusion

In this paper we describe the steps required to resize a well known molecular dynamics application and evaluate resulting performance benefits. We present a case study using LAMMPS and identify the minor modifications required to execute it with ReSHAPE. Experimental results show that dynamic resizing significantly improves LAMMPS execution turn-around time as well as overall cluster utilization under typical workloads.

Although data redistribution using file-based checkpointing is expensive (compared with message-passing based redistribution available for some common distributed data structures [3]), we find that it is a realistic approach for production scientific codes such as LAMMPS. This approach takes advantage of checkpoint and recovery mechanisms already available in the code; deep understanding of distributed data structures is not required. The extra overhead is relatively modest for long-running applications as long as resizing is not done too often, and in fact, may not be any added cost at all since the usual (statically scheduled) case would write checkpoint files periodically anyway. We also note that this approach requires no changes to the ReSHAPE framework. The ReSHAPE runtime environment treats resizable LAMMPS jobs like any other resizable job, irrespective of how data redistribution is accomplished.

References

1. Sudarsan, R., Ribbens, C.: ReSHAPE: A Framework for Dynamic Resizing and Scheduling of Homogeneous Applications in a Parallel Environment. In: Proceedings of the 2007 International Conference on Parallel Processing, Xian, China, pp. 44–54 (2007)

2. Sudarsan, R., Ribbens, C.: Scheduling Resizable Parallel Applications. In: Proceedings of 23rd IEEE International Parallel and Distributed Processing Symposium (to appear)
3. Sudarsan, R., Ribbens, C.: Efficient Multidimensional Data Redistribution for Resizable Parallel Computations. In: Proceedings of the International Symposium of Parallel and Distributed Processing and Applications, Niagara falls, ON, Canada, pp. 182–194 (2007)
4. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. Journal of Computational Physics 117, 1–19 (1995)
5. Plimpton, S., Pollock, R., Stevens, M.: Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In: Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, pp. 8–21 (1997)
6. Farkas, D., Mohanty, S., Monk, J.: Linear grain growth kinetics and rotation in nanocrystalline ni. Physical Review Letters 98, 165502 (2007)
7. Huedo, E., Montero, R., Llorente, I.: A Framework for Adaptive Execution in Grids. Software Practice and Experience 34, 631–651 (2004)
8. Buisson, J., Sonmez, O., Mohamed, H., Lammers, W., Epema, D.: Scheduling malleable applications in multicluster systems. In: Proceedings of the 2007 IEEE International Conference on Cluster Computing, Austin, USA, pp. 372–381 (2007)
9. Vadhiyar, S., Dongarra, J.: SRS - A framework for developing malleable and migratable parallel applications for distributed systems. Parallel Processing Letters 13, 291–312 (2003)
10. Moriera, J.E., Naik, V.K.: Dynamic resource management on distributed systems using reconfigurable applications. IBM Journal of Research and Development 41, 303–330 (1997)
11. Cirne, W., Berman, F.: Using Moldability to Improve the Performance of Supercomputer Jobs. Journal of Parallel and Distributed Computing 62, 1571–1602 (2002)
12. Open MPI v1.3 (2009), http://www.open-mpi.org
13. Van der Wijngaart, R., Wong, P.: NAS Parallel Benchmarks Version 2.4. NASA Ames Research Center: NAS Technical Report NAS-02-007 (2002)
14. Daw, M.S., Baskes, M.I.: Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. Phys. Rev. B 29, 6443–6453 (1984)

Performance Evaluation of Collective Write Algorithms in MPI I/O

Mohamad Chaarawi, Suneet Chandok, and Edgar Gabriel

Parallel Software Technologies Laboratory,
Department of Computer Science, University of Houston
{mschaara,schandok,gabriel}@cs.uh.edu

Abstract. MPI is the *de-facto* standard for message passing in parallel scientific applications. MPI-IO is a part of the MPI-2 specification defining file I/O operations in the MPI world. MPI-IO enables performance optimizations for collective file I/O operations as it acts as a portability layer between the application and the file system. The goal of this study is to optimize collective file I/O operations. Three different algorithms for performing collective I/O operations have been developed, implemented, and evaluated on a PVFS2 file system and over NFS. The results indicate that different algorithms promise the highest write bandwidth for different number of processors, application settings and file systems, making a one-size-fits-all solution inefficient.

1 Introduction

Many scientific applications utilizing parallel computers have to analyze tremendous amounts of data. The main challenge for such applications is the limited performance of individual magnetic hard drives, respectively of the entire I/O subsystem attached to typical clusters. Compared to the performance of CPU, memory and networking cards, a magnetic hard drive offers multiple orders of magnitude of higher latencies and lower bandwidths. Operating Systems try to hide the latency of file I/O operations by applying buffering and caching techniques, which show significant performance improvements for certain (regular) scenarios, but lead to performance degradation for applications having more irregular I/O patterns. In order to overcome the bandwidth limitations, most systems combine multiple disks to a single logical unit in a RAID configuration [1], such that files can be striped over multiple disks. The I/O performance of an application will thus depend on characteristics of the storage device, the file system utilized, the network interconnect used between compute nodes and the storage as well as in-between the compute nodes, and the I/O pattern of the applications.

The MPI 2 specification [2] includes routines for handling files in a parallel application, leveraging existing concepts from MPI 1, such as process groups and derived data-types. Among the most important features of the MPI I/O specification is the notion of a file view, which defines the portion of a file accessible to a particular process. Declaring a file view allows the MPI I/O implementation to pre-calculate offsets for the subsequent I/O operations from a process, detect overlap or the lack of overlap between individual processes, and potentially prefetch data items for read operations.

The MPI I/O interfaces offer both blocking and non-blocking operations to access a file, as well as the notion of individual vs. collective I/O operations. The latter offers the possibility to concatenate data from multiple processes, potentially avoiding a sequence of small, individual file requests, but post a single, large I/O request instead. ROMIO [3], the most wide-spread implementation of MPI I/O as of today, utilizes so-called two-phase collective I/O operations [4], which combines the data and posts I/O requests similarly to what we described in this paragraph. This approach has also been taken by some file systems to accumulate multiple user-level requests [5,6], and has been extended in various ways, e.g. to reduce the amount of meta-data required to be communicated between the processes by using derived data types [7] instead of lists of offsets. In [8], an adaptive approach for parameters that are passed as file hints is presented to optimize the performance of collective I/O operations on the SX vector computer with the GFS [9] file system. Yu et. al. exploit the file joining feature of Lustre to optimized collective write operations [10].

In this paper we analyze the performance characteristics of three different algorithms to implement collective write operations. Although the algorithms here can easily be transformed and used for read operations as well, we stick with write operations due to space limitations. The first algorithm is based on the two phase collective I/O algorithm described above, having the option to group internally the processes and thus vary the number of processes executing file I/O operations. The second is a modification of the two-phase collective I/O algorithm which does not optimize the file access to the hard drive, but the communication occurring during the shuffle operation. Lastly, the third algorithm avoids any communication between the processes and has each process handle its own I/O requests. We explore the performance behavior of these algorithms over a PVFS2 file system and over NFS, and compare them to the performance numbers achieved using ROMIO.

The remainder of the paper is organized as follows: section 2 gives details to the three algorithms explored in this paper. In section 3 we present the results obtained over PVFS2 and NFS. Finally, in section 4 we summarize this work and outline the future work in this area.

2 Collective Write Algorithms

This section describes the algorithms that have been implemented within this study. Two of the algorithm are derived from the two-phase collective I/O approach used in ROMIO, while the third algorithm acts similarly to individual I/O routines. The routines have been implemented in a stand-alone library utilizing the profiling interface of MPI to intercept the `MPI_Init` and the `MPI_Finalize` functions. The library implements a subset of the MPI I/O routines defined in the standard. In `MPI_Init` the library reads a configuration file which specifies the algorithm to be used for collective I/O operations. This allows us to test various algorithms without having to recompile the library.

For simplicity of low level I/O operations, a file I/O layer is introduced consisting of four simple interfaces, namely open, read, write and close. Two different implementations for the lower level I/O operations are available as of today. The synchronous low level I/O functions use `pwrite/pread` while the asynchronous version relies on the `aio_write/aio_read` [11] operations. Both sets of routines eliminate the need for seek operations, since the user has to provide explicitly the offset into the file for every operation. Since our results did not show a significant impact on using the synchronous or asynchronous low-level I/O functions for the platforms tested, we stick for the sake of simplicity to the synchronous low-level I/O operations for the rest of the paper.

2.1 Dynamic Segmentation Algorithm with Multiple Writers

This algorithm follows mostly the two-phase collective I/O operations outlined in the introduction. Its main goal is to combine data from multiple processes in order to minimize the number of I/O operations presented to the file system and avoid rewinds on the disk if possible. In a first step, all processes share location information about the data to be written with each other. Using two `MPI_Allgather()` operations, all processes share the list of file offsets and number of elements to be written with each other within the given collective write operation. Thus, every process has the knowledge of the operations to be performed by every other process. All processes can than sort these lists in an ascending order of the file offsets. The lists are divided in cycles of operation, where in each cycle, a fixed number of bytes are written to disk. A process can calculate how many elements it has to contribute for the write operation in that cycle. Using an extended version of the `MPI_Gatherv()` function, each process sends its elements contributing in the current cycle to the writer process assigned to him.

Note, that there can be more than one writer process, depending on the number of writer processes defined in the configuration file. Each writer process would handle the I/O for a certain number of processes. For example, for 24 processes and 4 writers, the processes would be grouped such that processes $0 - 5$ will have the rank 0 as a writer, processes $6 - 11$ process 6 as a writer, processes $12 - 17$ process 12 as a writer and so on. The writers would gather all the information and data needed from the other processes in their group and perform the actual write to disk.

Figure 1 shows a case where three processes are writing collectively with a cycle size of five bytes using a single writer process, namely rank zero. It shows that the operation is performing sequential disk access under optimal conditions with each process contributing varying amount of data in each cycle. Furthermore, it highlights one of the important characteristics of this algorithm, namely the fact, that a process might not be actively involved in every cycle of this algorithm. Thus, while the algorithm optimizes the disk access operations, the communication pattern deployed is suboptimal due to the fact that the data

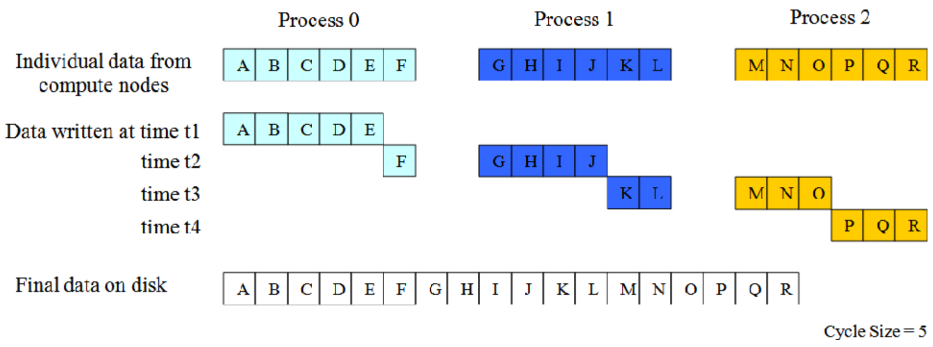


Fig. 1. Sketch of the write dynamic segmentation algorithm with a single writer

contributed by a particular process varies per cycle and thus does not make use of all communication channels in each cycle. The communication costs are affected however positively when using multiple writers in this algorithm, since it leads to the forming of smaller subgroups of process, which each execute internally a gather operation. Thus, the congestion typically occurring at the root process of the gather operation is avoided by having multiple root processes.

Since large write-all operations are typically executed in multiple cycles, one of the open questions is whether the size of the temporary buffer used for concatenating data from multiple processes, in the following referred to as the cycle buffer, shall be fixed and independent of the number of writer processes, or whether it should scale with the number of writers. We will evaluate both options in the subsequent results section.

2.2 Static Segmentation Algorithm

In this algorithm, data is gathered from all processes at a root process which will perform the low-level write operation. Data is written in fixed chunks, with the size of the chunk being a parameter of the configuration file read in `MPI_Init`. In contrary to the previous algorithm, the root process gathers a fixed number of bytes from all processes in each cycle. Thus, the algorithm does not necessarily reduce the overall number of I/O requests presented to the file system, but reduces the number of processes executing these I/O requests. Furthermore, due to the fact that every process contributes in every cycle a constant amount of data, this algorithm makes a better use of the communication resources in the cluster.

Figure 2 shows a scenario where three processes are writing collectively with cycle buffer size of two bytes. It shows that the operation is performed in three cycles. After the 1st cycle, the file pointer needs to be moved back for the next cycle. Note, that the algorithm does not support as of today the notion of multiple writers, i.e. one process is only allowed to execute I/O operations. Similarly to the dynamic segmentation algorithm, the question on whether to scale the cycle buffer with the number of processes or whether to keep it constant independently of the number of process used, arises.

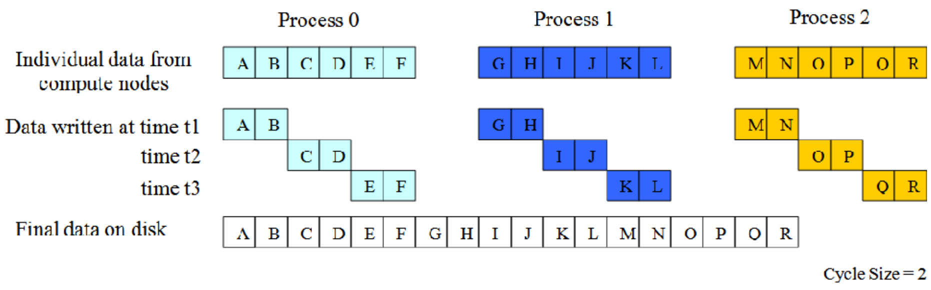


Fig. 2. Sketch of the write static segmentation algorithm

At first sight, this algorithm seems counterintuitive to the common knowledge which states, that file access operations are the most time consuming part of collective I/O operations. However, many large scale installations provide huge caches on the I/O nodes, which effectively decouple the compute cluster form the storage devices, and thus show - from the application perspective - virtually no sensitivity to irregular or strided file access patterns [12]. Furthermore, one of the distinctive features of a technology currently on the rise, solid state hard drives (SSD), is its insensitivity to irregular access in the file. For these two scenarios, the static segmentation algorithm optimizes the second most time consuming operation, namely the communication occurring during the shuffle step.

2.3 Individual Write

This algorithm avoids communication operations entirely and has each process write its data individually to the hard drive. The `MPI_File_write_all` operation exposes therefore the behavior that the application would face when using individual `MPI_File_write` operations instead of the collective version. The main difference compared to the latter approach is that the collective operation internally structures the I/O operations in cycles, similarly to the previous algorithms, in order to overlap the I/O operations with the calculation of the file offsets based on the file view and the merging of different segments.

Note, that this algorithm has also been extended by using a scheduling approach to control the number of processes concurrently performing I/O operations, and thus limit the burden on meta-data servers for some file systems. Due to space limitations we skip however this (fourth) algorithm.

3 Performance Evaluation

This section presents the performance evaluation of the three algorithms presented previously. For the dynamic segmentation algorithm using multiple writers, we explore the performance using 1, 2, 4, 8, 12, and 24 writers. We also compare these algorithms to the performance of ROMIOs `MPI_File_write_all` routine. The ROMIO algorithm has been executed without passing any additional hints to the library, since the main goal of using ROMIOs implementation within this context is to have a baseline for the comparison of our algorithms. The cluster used for these tests consists of 24 single process dual-core AMD Opteron nodes and 5 dual-processor quad-core AMD Opteron nodes, providing a total of 88 compute cores. The nodes are connected by a 4x InfiniBand network interconnect. It has a parallel file system (PVFS2) mounted as '/pvfs2', which utilizes 22 hard drives, each hard drive is located on a separate compute node. The PVFS2 file system internally uses the Gigabit Ethernet network to communicate between the pvfs2-servers. The home file system on is NFS mounted from the front-end node.

We executed a simple benchmark in which processes collectively write *max_size* bytes of data for a given number of iterations to the file. The file view is determined by another parameter (*segment_size*), which allows to control the size of the data portion 'owned' by a process. To explain the correlation between the two parameters, consider an example, where *max_size* is set to 8 bytes, while *segment_size* is set to 2 bytes. In a four process test case, a call to `MPI_File_write_all` having to write *max_size* bytes of data per process would lead to process 0 having to write two bytes each at offsets (0,8,16,24), process 1 writing two bytes each at the offset (2,10,18,26) and so on. We measure the execution time of the test required to write **all** data to file, and calculate the bandwidth achieved by taking the size of the overall file created and overall execution time. Each test has been executed three times, and we present here the maximum bandwidth achieved through these runs.

3.1 Results Achieved over PVFS2

When writing over PVFS2, each process executes `MPI_File_write_all` operations writing 20MB of data per function call, writing all-in-all 1GB of data to file. Note, that we did perform tests with even larger files, which lead however to the same performance numbers that we present in this paper. Thus, the overall file size for the 24 processes test cases is 24GB and for the 48 processes test cases is 48 GB. Furthermore, we vary the segment size (2MB, 10MB, 20MB) and the cycle buffer size (1MB, 10MB, 20MB) for our tests.

The results of the first set of tests executed over PVFS2 are shown in Fig. 3. For both, the 24 and the 48 processes tests, the dynamic segmentation and the static segmentation algorithm were using scaling cycle buffers, as explained in the according subsections. Both graphs have in common, that the dynamic segmentation algorithm with 24 writers and the individual algorithm achieve the highest bandwidth. More generally, as the number of writers increase in the dynamic segmentation algorithm, so does the bandwidth achieved for those operations. The static-segmentation algorithm shows a bad performance in these tests. ROMIO is achieving a reasonably good performance, although the two top performing algorithms are significantly outperforming the default ROMIO version.

All algorithms show increasing performance with increasing segment sizes. The main reason for this is that the number of data blocks that have to be sorted and potentially merged is decreasing, since the size of each data block is

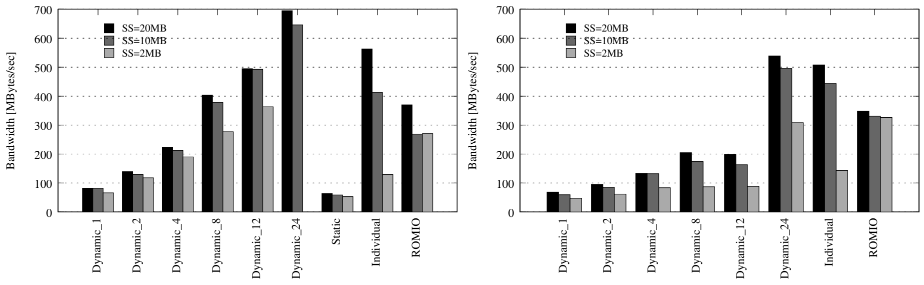


Fig. 3. Performance Comparison for 24 processes (left) and 48 processes (right) with varying the segment size and keeping the cycle buffer size constant at 20MB

Table 1. Performance Breakdown (in seconds)of the Dynamic Segmentation and Individual Algorithms over PVFS2 with 24 and 48 processes with segment size and cycle buffer size being 20 MB

	Gathering		I/O		Total	
	24	48	24	48	24	48
Dynamic_24	0.74	2.97	33.18	99.39	34.91	104
Dynamic_12	1.8	5.24	46.62	232.28	50.08	239.63
Dynamic_1	25.88	49.46	285.14	755.75	325.74	837.83
Individual	N/A	N/A	39.8	86.49	39.8	86.49

increasing. ROMIO shows a significantly lower sensitivity to the segment size than our algorithms, probably due to the usage of derived data types [7] instead of lists of offsets for the according operations.

Comparing the performance of the 24 and 48 process test-cases, it is notable, that the overall bandwidth of all algorithms drops when using two processes per node. Tabel 1 shows the performance breakdown of the dynamic segmentation and individual algorithms. Both cases (24 and 48 processes) show that most of the time is spent in I/O operations, and a smaller fraction is spent on the data gathering operations. However, the data also indicates, that although the data volume doubles between the two cases, both the I/O and the communication costs increase more severely, e.g. by a factor of 2-5. Note that for the 48 processes case, we were not able to get a result for the static segmentation algorithm due to the enormous amount of temporary buffer required for that scenario (960MB). With a fixed cycle buffer size, the algorithm could finished successfully, but performed poorly overall.

In Fig. 4, we analyze the effect of the cycle buffer size on the algorithms. For this, we execute the same write tests keeping the segment size constant at 20MB. The results show that increasing the cycle buffer size improves the performance but not by a huge factor. The individual algorithm is an exception there, where increasing the cycle buffer size does not necessarily yield a better performance.

Finally, table 2 shows the difference between fixed and scaling cycle buffer sizes when using the dynamic segmentation algorithm. The results show that

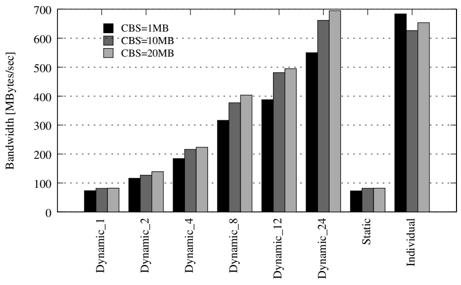


Fig. 4. Comparing the performance of some of the algorithms while varying the cycle buffer size and keeping the segment size constant at 20MB

Table 2. Bandwidth comparison (MB/sec) of the Dynamic Segmentation algorithm with Constant vs. scaling cycle buffer size of 20 MB and a segment size of 20 MB

	24 procs		48 procs	
	constant	scaling	constant	scaling
Dynamic_1	74.47	82.27	55.09	68.7
Dynamic_2	116.52	139.31	86.87	95.35
Dynamic_4	189.37	223.48	112.72	133.59
Dynamic_8	328.27	403.49	110.21	204.81
Dynamic_12	330.99	494.89	71.59	198.53
Dynamic_24	350.41	694.82	280.26	538.81

having each writer write the specified cycle buffer size in a cycle would be better than dividing the cycle buffer size over all the writers in each cycle. The reason would be that with the scaling cycle buffer size, the actual I/O is done with larger sequential chunks of data as compared with small chunks of the fixed cycle buffer size. Another common observation between the two approaches is that increasing the number of writers over PVFS2 still provides better results.

3.2 Results Achieved over NFS

Although NFS is considered to not be well suited for parallel I/O, it is as of today the most wide spread file system on small and medium size clusters. End-users experimenting with MPI I/O over NFS should not see a performance degradation compared to sequential POSIX I/O even on this file system, since this will discourage them from using MPI I/O in their codes.

In order to keep the execution time of our tests within a reasonable time frame, we set each process to write only 100MB of data. Tests have been executed with various cycle buffer size (1MB, 10MB, 20MB), and keep the segment size constant at 2MB.

The results that were gathered over NFS show significant deviations from the PVFS2 results. For the dynamic segmentation algorithm, the lower the number

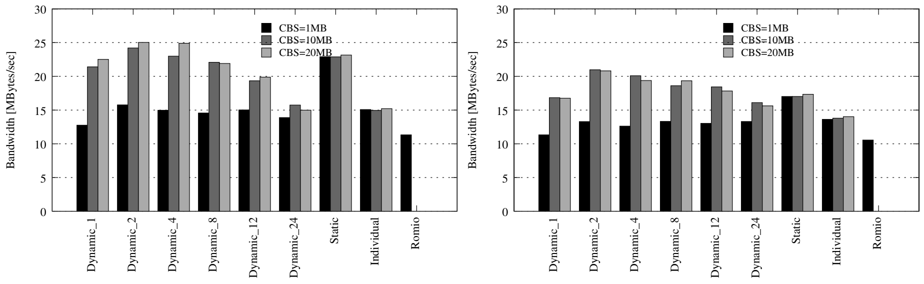


Fig. 5. Performance Comparison for 24 processes (left) and 48 processes (right) with varying the cycle buffer size and keeping the segment size constant at 2MB

of writers, the better the performance. The static segmentation algorithm using scaling cycle buffers performs well in that case, compared to the other algorithms. Most algorithms that we tested outperformed ROMIOs version in this setting. Since the cycle buffer size is not relevant for ROMIO, the graph shows only one bar for ROMIO. Independent of the algorithm used, the user still would observe a performance hit when using MPI I/O over NFS compared to the raw write performance of a single hard drive (35 MB/s sustained).

4 Summary

In this paper, we described three algorithms for MPI-IO collective write operations, the dynamic segmentation, static segmentation, and individual algorithms. The testing was done over two file systems, PVFS2 and NFS. The results show that there is a large room for optimizations within the collective I/O operations. The performance of the algorithms depended on the file system, number of processes and the file view (segment size) utilized by the processes, and lead in fact to different algorithms delivering the best performance.

Future work in this area includes further extending some of the algorithms described above, e.g. including the ability to have multiple writers for the static segmentation algorithms, and extend the individual algorithms by sophisticated scheduling approaches in between the processes to limit the burden on meta-data servers. Furthermore, we plan to evaluate the performance of the algorithms on a wide variety of hardware and software configurations in collaboration with various institutions, including a RAID of SSD disks. Preliminary tests on a Lustre file system have already been performed and revealed again a highly different behavior of the algorithms. The long term goal of the project is to develop a flexible module for collective I/O operations that can easily adjust to various hardware and software configurations and choose the right algorithm dynamically, using dynamic runtime adaption techniques.

Acknowledgments. This research was funded in part by a gift from the Silicon Valley Community Foundation, on behalf of the Cisco Collaborative Research Initiative of Cisco Systems.

References

1. May, J.: Parallel I/O for High Performance Computing. Morgan Kaufmann, San Francisco (2001)
2. Message Passing Interface Forum: MPI-2: Extensions to the Message Passing Interface (July 1997), `http://www.mpi-forum.org`
3. Thakur, R., Gropp, W., Lusk, E.: On Implementing MPI-IO Portably and with High Performance. In: Proceedings of the sixth workshop on I/O in parallel and distributed systems, pp. 23–32 (1999)
4. Thakur, R., Gropp, W., Lusk, E.: Data Sieving and Collective I/O in ROMIO. In: FRONTIERS 1999: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation, p. 182. IEEE Computer Society, Los Alamitos (1999)
5. Prost, J.P., Treumann, R., Hedges, R., Jia, B., Koniges, A.: MPI-IO/GPFS, an Optimized Implementation of MPI-IO on top of GPFS. In: Proceedings of Supercomputing (November 2001)
6. Isaila, F., Malpohl, G., Olaru, V., Szeder, G., Tichy, W.: Integrating collective I/O and cooperative caching into the "clusterfile" parallel file system. In: Proceedings of the 18th Annual International Conference on Supercomputing, Sain-Malo, France, pp. 58–67. ACM Press, New York (2004)
7. Ching, A., Choudhary, A., Liao, W.K., Ross, R., Gropp, W.: Efficient structured data access in parallel file systems. In: Proceedings of the IEEE International Conference on Cluster Computing (December 2003)
8. Worringen, J.: Self-adaptive Hints for Collective I/O. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 202–211. Springer, Heidelberg (2006)
9. Ohtani, A., Aono, H., Tomaru, H.: A File Sharing Method for Storage Area Network and Its Performance Verification. NEC Res. Dev. 44(1), 85–90 (2003)
10. Yu, W., Vetter, J., Canon, R.S., Jiang, S.: Exploiting lustre file joining for effective collective io. In: CCGRID 2007: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, pp. 267–274. IEEE Computer Society Press, Los Alamitos (2007)
11. Wadleigh, K.R., Crawford, I.L.: Software Optimization for High-Performance Computing. Prentice-Hall, Englewood Cliffs (2000)
12. Simms, S.C., Pike, G.G., Balog, D.: Wide Area Filesystem Performance using Lustre on the TeraGrid. In: Teragrid Conference (2007), `http://datacapacitor.researchtechnologies.uits.iu.edu/lustre_wan_tg07.pdf`

A Scalable Non-blocking Multicast Scheme for Distributed DAG Scheduling[⋆]

Fengguang Song[1], Jack Dongarra[1,2], and Shirley Moore[1]

[1] University of Tennessee, EECS Department
Knoxville, Tennessee, USA
{song,dongarra,shirley}@eecs.utk.edu
[2] Oak Ridge National Laboratory,
Oak Ridge, Tennessee, USA

Abstract. This paper presents an application-level non-blocking multicast scheme for dynamic DAG scheduling on large-scale distributed-memory systems. The multicast scheme takes into account both network topology and space requirement of routing tables to achieve scalability. Specifically, we prove that the scheme is deadlock-free and takes at most $logN$ steps to complete. The routing table chooses appropriate neighbors to store based on topology IDs and has a small space of $O(logN)$. Although built upon MPI point-to-point operations, the experimental results show that our scheme is significantly better than the simple flat-tree method and is comparable to vendor's collective MPI operations.

1 Introduction and Motivations

Multicore architectures are capable of running many threads simultaneously and require future parallel software be fine-grained and asynchronous [1,2,3]. An approach to developing scalable parallel software is to place fine-grained computations in a directed acyclic graph (DAG) and schedule them dynamically (data-driven or demand-driven) [4]. Although a centralized DAG scheduler works well on SMPs, it will not scale well on systems with thousands of nodes or tens of thousands of cores. One way to overcome the scalability problem is to adopt a decentralized scheduler, that is, each node runs a private DAG scheduler and communicates with other nodes regarding data dependences only when necessary. Ideally, the distributed scheduler has no globally shared data structures, no requirement of much space to store DAGs, and no blocking operations. Furthermore, it respects the critical path, takes into account data locality, maintains load balancing, and performs communication efficiently.

Instead of solving all the problems at once, we study how to perform communication efficiently during dynamic DAG scheduling. The most common communication operation used to execute DAGs is multicasting where a completed task must notify its descendants that are blocked awaiting its output (Fig. 1).

⋆ This material is based upon work supported by the Department of Energy Office of Science under grant No. DE-FC02-06ER25761 and by Microsoft Research.

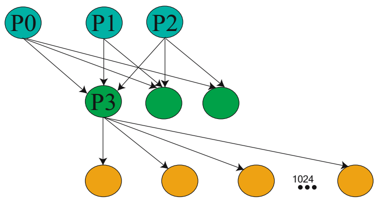


Fig. 1. Data multicast from parent to children in a DAG

MPI libraries provide users with optimized broadcast operations on nearly all high performance machines. Due to the dynamic and irregular parent/children relationship in general DAGs, there could be 2^N possible subsets of processes for broadcasting, where N is the number of processes. For every finished task and its corresponding children, one has to call `MPI_Comm_create()` followed by `MPI_Bcast` to realize the multicast. Even if we ignore the time to create the communication groups, multiple MPI broadcasts involving the same process have to be executed in sequence because MPI broadcast is a collective operation. Figure 1 shows an example where P3 is involved in three communication groups with broadcast roots $P0$, $P1$, $P2$, respectively.

This paper presents a novel multicast scheme to enable dynamic DAG scheduling on large-scale distributed systems with tens of thousands of processors. The multicast scheme is non-blocking, topology-aware, scalable, and deadlock-free, and it supports multiple concurrent multicasts. We compare its performance to a flat-tree multicast and a vendor `MPI_Bcast`. Based on the experimental results, our multicasting scheme is significantly better than the simple flat-tree method and comparable to the optimized collective MPI broadcast.

2 Computation Model

2.1 Symbolic Task Graph

We represent the semantics of programs with loop nest control structures by polyhedrons such that each task instance corresponds to a unique *coordinate* or *iteration vector*. A task instance is denoted by a tuple `(type, iteration vector)`. By identifying data dependences between tasks, we are able to construct the whole DAG. Similar to the method introduced by [5], we define a task graph symbolically as follows: $G = \langle T, E \rangle$, where

$$T = \{\text{task t } : t = (type, \boldsymbol{u})\}.$$

The set of edges are defined by a set F of symbolic functions:

$$F = \{f_i \text{ for certain task type } i\} \text{ and } f_i : \{\boldsymbol{u}\} \to T.$$

Given a task instance $(t, \boldsymbol{u})$, $f_t(\boldsymbol{u})$ generates a set of tasks that are dependent on task $(t, \boldsymbol{u})$.

2.2 Programming Model

We designed a simple application programming interface (API) and implemented a runtime system prototype to support dynamic DAG scheduling. After the user implements the API routines, the underlying runtime system can automatically parallelize and execute the DAG on shared- or distributed-memory systems. The ANSI C programming interface routines are listed below:

```
int  get_children(const Task t, Task children);
int  get_num_parents(const Task t);
void set_entry_task(const Task t);
void set_exit_task(const Task t);
```

Note that we can obtain the child tasks easily by calling `get_children()` if the finished parent task is given. As long as each member of the multicast group is notified of the parent task, it is able to deduce the whole group immediately. If the `get_children()` function is not feasible, the group members have to be included explicitly in messages.

3 Multicast Scheme Overview

When a set of processes are executing a DAG, multiple sources may want to notify different groups of children simultaneously. The new multicast method is able to provide this functionality automatically. The multicast scheme is essentially an application-level routing method. Every process owns a compact routing table. Although each process only has knowledge of a few neighbors, the whole group of processes is represented by a collection of hierarchical trees. Most importantly, every process is the root node of its own multicast tree. The routing algorithm simply follows the tree to multicast data to a set of children.

In Fig. 2, the process on node `001` wants to multicast data to $\{010, 100, 101\}$. For this system with eight nodes, it takes three steps to complete the multicast. There could be at most eight processes running (at leaf nodes) on the system, but certain processes will be mapped to serve as "virtual masters" responsible for their corresponding subtrees. Our method to build the routing table guarantees that there exists a path from the source to every destination by filling in "virtual masters" (Lemma 1 in Sect. 7). The path length is bounded by $logN$.

4 Topology ID

To improve communication performance, it is critical to know the communication cost between a node and the other nodes on a system. One could build a $N \times N$ table to describe the latency and bandwidth information between every pair of nodes. But for a system with millions of nodes, it is too costly to build and maintain such a big table. Another natural approach is to use hierarchy as an abstraction to achieve scalability. The hierarchy abstraction has been widely used on the Internet, for example for DNS and IP addresses, as well as for message passing operations

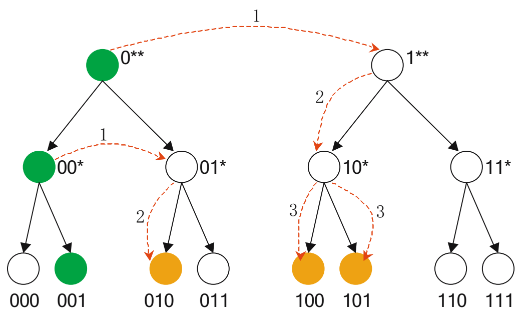


Fig. 2. Data multicasting in a data flow graph

on computational Grids. Instead of exposing all the other nodes to the source, the hierarchy technique utilizes a hierarchical tree to send data level by level. This way each node only communicates with a small number of nodes so that the system keeps scaling. We assign each node a topology ID on the system. Working like ZIP codes, we assume the longer the common prefix of the two nodes' topology IDs, the closer they are and the smaller the latency. When a process is running on a node with topology ID x, we say the process has topology ID x.

5 Extention to the Plaxton Neighbor Table

In this section, we briefly describe the Plaxton neighbor table and our extension to support multicasting. Plaxton uses an incremental routing approach similar to hypercube routing which resolves the destination node address dimension by dimension [6]. Supposes a system has $n = 2^m$ nodes, where m is a multiple of b. Plaxton assumes that each node has a label which is independently and uniformly distributed at random between 0 and $n-1$. Instead of using a random label for each node, we assign a topology ID $tid \in [0...n-1]$ to each node. The topology ID reflects the latency relationship (near or far) between two nodes, and is expressed as a sequence of $\frac{m}{b}$ digits with the base 2^b. For instance, if one system has $4096 = 2^{12}$ nodes, base $= 2^3$ leads to a 4-digit octal topology ID.

Every node has its own neighbor table T. Table T consists of r rows and c columns, where r is equal to the number of digits $(=\frac{m}{b})$ and c is equal to the digit's base $(= 2^b)$. Table entry $T[i,j]$ stores the forwarding node address. In the case of application-level multicasting, we store an MPI rank as a forwarding address. Let node x have a topology ID of $id^{(x)} = d_0d_1 \dots d_{r-1}$. If table entry $T[i,j]$ in node x contains node y which has topology ID $id^{(y)}$, then $id^{(x)}$ and $id^{(y)}$ must satisfy the following two conditions:

(1) $id_0^{(x)}id_1^{(x)} \dots id_{i-1}^{(x)} = id_0^{(y)}id_1^{(y)} \dots id_{i-1}^{(y)} = d_0d_1 \dots d_{i-1}$,
(2) $id_i^{(x)} \neq j$ and $id_i^{(y)} = j$.

Please note that there could exist a set Y of nodes meeting the above conditions for node x. For instance, table entry $T[3,5]$ in a node with octal topology ID

012345 can contain any node with a topology ID $\in 012[\{0-7\}/3\}][0-7]^+$. Therefore a decision function is needed to choose the best candidate. For instance, Plaxton chooses $y^* = Min_{y \in Y} CommCost(x, y)$ as the best neighbor.

In the case of $Y = \emptyset$, Plaxton assumes an ordering in the set of n nodes and picks a node y that matches node x in the suffix $i, i+1, \ldots, r-1$ digits with the highest order. In contrast to the full Plaxton neighbor table, we leave those entries empty and prove this modification avoids cycles in application-level multicasting (Theorem 2 in Sect. 7).

5.1 Compact Routing Table

While routing tables are usually used to connect nodes, we use them to connect processors and processes in the context of application-level multicasting. Assume a system has n processors and the base of topology IDs is equal to c, then the routing table will have $\frac{log_2(n)}{log_2(c)}$ rows and c columns. The routing table occupies a small amount of space even for large-scale systems. If a system has one million (2^{20}) cores, a base of 16 results in a routing table of 5 rows by 16 columns that equals 80 entries. If one has a billion (2^{30}) cores, the routing table is of 6 rows and 32 columns given the base 32. Every table entry just stores a single integer.

6 Algorithms

This section describes how every process builds its own routing table when the application first starts and how the process constantly receives messages and forwards them to proper destinations.

6.1 Building a Local Routing Table

Before doing any real work, each MPI process first builds a local routing table. Each process's topology ID is assigned by users based on the network topology. In Fig. 3, a process scans every other process ID and compares that process's topology ID to its own topology ID to fill in the routing table. When there are multiple processes that are legitimate to be stored in T[i,j], we either pick a process randomly or find the closest process. In our experiments on Myrinet, the random method is slightly better than the nearest neighbor method.

6.2 Forwarding Algorithm

While participating in the multicast, a process works as either an internal node or a leaf in the multicast tree. Whenever the root is given, the locations of receivers become fixed in the particular multicast tree. Data will always flow from root to leaves. Figure 3 shows how to find the next level of tree nodes in the multicast tree to which to forward. The index of the next level (i.e., `stage`) should be at least one level further from the root. The program looks up the table and gets a forwarding process for each child and stores it in array `destinations`.

```
typedef struct {
  int  table[NUM_LEVELS * NUM_COLS];
  int  topo_ids[MAX_NUM_PROCESSES];
}* NeighborTable;

int *candidates[NUM_LEVELS * NUM_COLS];
NeighborTable my_tbl;

for(p = 0; p < nprocs; p++) {
  if(p  == my_pid) continue;
  topid  = my_tbl->topo_ids[p];
  level  = longest_prefix(my_top_id, topid);
  column = get_kth_digit(topid, level);
  idx    = level * NUM_COLS + column;
  candidates[idx][counters[idx]++] = p;
}

/*choose proper neighbor from candidates*/
choose_best_neighbor(my_tbl, candidates);
```

```
while(1) {
  ...
  Received a message from process prev_topid;
  stage = longest_prefix(my_top_id, prev_topid)+1;
  for(i = 0; i < num_children; i++) {
    p = get_children(i);
    if(p == my_pid) continue;
    top_id = my_tbl->topo_ids[p];
    lcl = longest_prefix(my_top_id, top_id);
    if( lcl  >= stage) {
      column  = get_kth_digit(top_id, lcl);
      forward = TBL_ENTRY(my_tbl, lcl, column);
      if(!is_element(forward, destinations)) {
        destinations[idx++] = forward;
      }
    }
  }
  Send message to processes in destinations[];
}
```

Fig. 3. Algorithms for the non-blocking multicast scheme

7 Theorems

Lemma 1. *Suppose process P_x has a topology ID x and needs to send data to process P_z with topology ID z. Then there always exists a process P_y stored in P_x's neighbor table such that P_x can forward data to P_y and $LCD(y, z) \geq LCD(x, z) + 1$.*

Proof. Let $LCD(i, j)$ compute the longest common prefix length of i and j. Suppose $i = LCD(x, z)$, P_x will forward data to a process with a topology ID of the form $x_0x_1 \ldots x_{i-1}z_i * \ldots *$. It is easy to see that at least z have the form. So one of the processes of $P_z \cup \{\text{processes with ID } x_0x_1 \ldots x_{i-1}z_i * \ldots *\}$ will get the forwarded data. Therefore such a process must exist. By definition of LCD, we know $x_0x_1 \ldots x_{i-1} = z_0z_1 \ldots z_{i-1}$ and $x_i \neq z_i$. Given i and z, the forwaring algorithm chooses the process stored in the ith row and the z_ith column. Any process located in $T[i, z_i]$ will be the target to which P_x forwards data and it must exist. WLOG, let it be P_y with topology ID y. Since P_y is in $T[i, z_i]$ of P_x's neighbor table, $y_0y_1 \ldots y_{i-1} = x_0x_1 \ldots x_{i-1} = z_0z_1 \ldots z_{i-1}$ and $y_i = z_i$. Therefore, $y_0y_1 \ldots y_i = z_0z_1 \ldots z_i$. In other words, $LCD(y, z) = i + 1 \geq LCD(x, z) + 1$.

Lemma 1 proves that the forwarding method is always successful even if there exist empty entries in the process's routing table. For every step of forwarding, the longest common prefix length to the destination increases by at least one.

Theorem 1 (Reachability). *It is always possible to route a message from process P_x to process P_z and it takes at most m steps to reach P_z. m is the number of digits in topology IDs.*

Proof. By Lemma 1, there $\exists P_y$ such that P_x can forward data to P_y and $LCD(y, z) \geq LCD(x, z) + 1$. Since topology ID z has m digits, it takes at most m steps to send data to P_z.

Theorem 2 (Deadlock-freedom). *The forwarding mechanism guarantees that there is no cycle during the forwarding process.*

Proof. Suppose process x_1 wants to send a message to x_p, but there is a cycle $x_1 \rightarrow x_2 \ldots \rightarrow x_k \rightarrow x_1$ formed before the message reaches x_p. If $LCD(x_1, x_p) = d_0 d_1 \ldots d_{l_1}$, then by Lemma 1,

$$LCD(x_2, x_p) = d_0 d_1 \ldots d_{l_1} \ldots d_{l_2}$$

$$LCD(x_3, x_p) = d_0 d_1 \ldots d_{l_1} \ldots d_{l_2} \ldots d_{l_3}$$

$$\ldots$$

$$LCD(x_k, x_p) = d_0 d_1 \ldots d_{l_1} \ldots d_{l_2} \ldots d_{l_3} \ldots d_{l_k}$$

$$LCD(x_1, x_p) = d_0 d_1 \ldots d_{l_1} \ldots d_{l_2} \ldots d_{l_3} \ldots d_{l_k} \ldots d_{l_{k+1}}$$

There is a contradiction if we compare the first $LCD(x_1, x_p)$ and the last $LCD(x_1, x_p)$. Therefore, the forwarding mechanism guarantees there is no cycle.

8 Related Work

MPICH-G2 uses depth to represent where an MPI process is located in a computational Grid [7]. The depths include levels of *wide area*, *local area*, *system area*, and *machine-specific area*. The topology table represented by *depths* and *colors* is a global table for the whole grid and needs to be accessible by every process. Our topology ID representation has a distributed compact table and requires much less space.

Plaxton introduces local neighbor tables at each node [6]. Our work is an extension to Plaxton's neighbor table where we build routing tables for MPI processes instead of nodes. Since a user's processes are always a subset of all the nodes on the system, we modify the table-building method to allow empty entries (or "holes") in the routing table. In addition, we design a multicast scheme based on the extended routing table. Wu [8] designs a deadlock-free prefix-based multicasting scheme for irregular networks. Each outgoing channel of a node is assigned a label. The multicast packet is first forwarded up to the root and then forwarded down to leaves. But the whole system is based on a single spanning tree. In our multicast method, every process has its own spanning tree. Panda proposes a Hierarchical Leader Based approach to support one-to-many multicasting [9]. The set of nodes are grouped into subsets explicitly so that each subset is represented by a leader. Banerjee et al. also uses a hierarchical clustering method to multicast the data stream to large receiver sets [10]. In contrast, our method builds routing tables to form hierarchies automatically (represented by spanning trees).

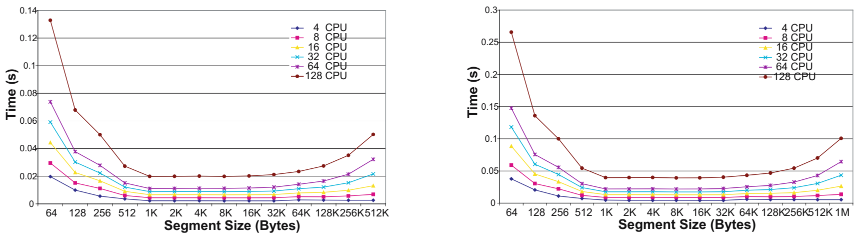
Bayeux uses the structure of Tapestry to provide an application-level multicast scheme for streaming multimedia applications [11,12]. Bayeux builds a distribution tree based on four control messages: *JOIN*, *LEAVE*, *TREE*, *PRUNE*. To construct a distribution tree, the source server must advertise the session information first. Then the clients have to join the session to form a tree. We don't need to construct distribution trees and simply use the implicit spanning trees to multicast data.

9 Experiments

We conducted experiments on a cluster machine with 64 nodes each with two processors. The cluster is connected by a Myrinet network. We also did experiments on a SGI Altix 3700 BX2 machine which has a fat tree network topology. The performance result on the SGI machine is similar to that on the cluster and is not shown here due to the space limitation.

9.1 Effect of Segments

The performance of the non-blocking multicast method could be affected by the segment size. Given a message size, we can choose to send it out once or in a number of segments. Figure 4 considers two message sizes: 512KB and 1MB. For each message size, we use different segments with sizes from 64Bytes to the whole message size and run it on a range of processors from 4 CPUs to 128 CPUs. Based on the data from Fig. 4, a segment size between 1KB and 32 KB always produces the best performance. Therefore in the experiments described in Sect. 9.2, we choose the segment size 8KB for our multicast method whenever possible.



(1) Multicast a 512KB message (2) Multicast a 1MB message

Fig. 4. Performance of multicast varies with different segment size on Myrinet

9.2 Experimental Results

We compare our non-blocking multicast method (labeled as "dag_mcast") to Myricom's MPICH-MX 1.1 MPI_Bcast (labeled as "mpi_bcast") and a straightforward implementation that uses a flat-tree to perform multicasting (labeled as "flat_mcast"). The flat-tree method simply sends the message to every destination one by one. Both flat_mcast and dag_mcast are implemented using point-to-point `MPI_Send` and `MPI_Recv` operations.

We conducted experiments on a range of processors from 16 up to 128. From Fig. 5, we can see that both `dag_mcast` and `mpi_bcast` are significantly better

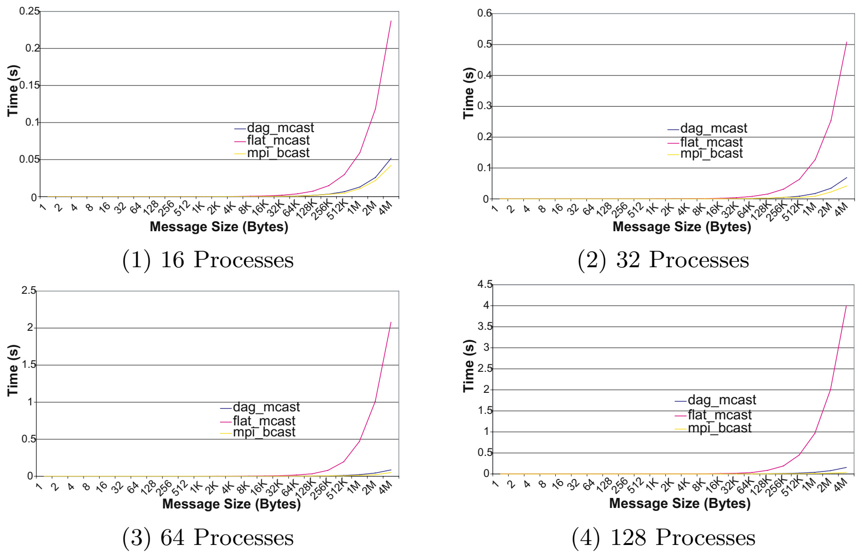


Fig. 5. Multicast performance on a cluster connected with Myrinet

than `flat_mcast`. And the non-blocking multicast method is comparable to the highly-optimized collective MPI_Bcast. Note that the time to invoke `MPI_Init` and `MPI_Comm_create` was not counted for the `mpi_bcast` experiments (in favor of `mpi_bcast`). The reason why the non-blocking multicast method is slower than MPI_Bcast is because our implementation is built over MPI point-to-point operations and we cannot do similar optimizations as MPI collective operations do (e.g., broadcast may be implemented as `scatter` followed by `allgather`, optimal binomial tree is built in advance). Although MPI_Bcast is faster, it is difficult to create communication groups and do collective broadcasts for every distinct group in dynamic DAG scheduling programs.

10 Conclusion

Our non-blocking multicast scheme is designed to support dynamic DAG scheduling on distributed-memory machines. While it is possible to use `MPI_Bcast` directly to implement it, creating communication groups and performing collective operations for arbitrary sets of parent/children is cumbersome to program. We have designed a multicast scheme, using topology IDs, compact routing tables, and multiple spanning trees. The multicast scheme is proven to be deadlock free, scalable in terms of time and space, topology-aware, and non-blocking. Our experimental results show that performance of our scheme is significantly better than the simple flat-tree method and comparable to vendor-optimized collective MPI operations.

References

1. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M.: Larrabee: a many-core x86 architecture for visual computing. ACM Trans. Graph. 27(3), 1–15 (2008)
2. Golla, R.: Niagara2: A highly threaded server-on-a-chip (2007)
3. Le, H.Q., Starke, W.J., Fields, J.S., O'Connell, F.P., Nguyen, D.Q., Ronchetti, B.J., Sauer, W.M., Schwarz, E.M., Vaden, M.T.: IBM Power6 microarchitecture. IBM J. Res. Dev. 51(6), 639–662 (2007)
4. Buttari, A., Langou, J., Kurzak, J., Dongarra, J.: A class of parallel tiled linear algebra algorithms for multicore architectures. Lapack working Note 191 (2007)
5. Cosnard, M., Jeannot, E.: Compact dag representation and its dynamic scheduling. J. Parallel Distrib. Comput. 58(3), 487–514 (1999)
6. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing nearby copies of replicated objects in a distributed environment. In: SPAA 1997: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures, pp. 311–320. ACM Press, New York (1997)
7. Karonis, N.T., Toonen, B., Foster, I.: MPICH-G2: A Grid-enabled implementation of the message passing interface. Journal of Parallel and Distributed Computing 63(5), 551–563 (2003); Special Issue on Computational Grids
8. Wu, J., Sheng, L.: Deadlock-free multicasting in irregular networks using prefix routing. The Journal of Supercomputing 31, 63–78 (2005)
9. Panda, D., Singal, S., Kesavan, R.: Multidestination message passing in wormhole k-ary n-cube networks with base routing conformed paths. IEEE Transactions on Parallel and Distributed Systems 10(1), 76–96 (1999)
10. Banerjee, S., Bhattacharjee, B., Kommareddy, C.: Scalable application layer multicast. In: SIGCOMM 2002: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 205–217. ACM, New York (2002)
11. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, Berkeley, CA, USA (2001)
12. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.D.: Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In: NOSSDAV 2001: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, pp. 11–20. ACM Press, New York (2001)

On the Origin of Grid Species: The Living Application

Derek Groen[1,2], Stefan Harfst[1,2], and Simon Portegies Zwart[1,2,3]

[1] Section Computational Science, University of Amsterdam,
Amsterdam, The Netherlands
djgroen@uva.nl

[2] Astronomical Institute "Anton Pannekoek", University of Amsterdam,
Amsterdam, the Netherlands

[3] Sterrewacht Leiden, Universiteit Leiden
Leiden, The Netherlands

Abstract. We present the living application, a method to autonomously manage applications on the grid. During its execution on the grid, the living application makes choices on the resources to use in order to complete its tasks. These choices can be based on the internal state, or on autonomously acquired knowledge from external sensors. By giving limited user capabilities to a living application, the living application is able to port itself from one resource topology to another. The application performs these actions at run-time without depending on users or external workflow tools. We have applied this new concept in a special case of a living application: the living simulation. Today, many simulations require a wide range of numerical solvers and run most efficiently if specialized nodes are matched to the solvers. The idea of the living simulation is that it decides itself which grid machines to use based on the numerical solver currently in use.

1 Introduction

A grid application consists of a range of tasks, each of which may run most efficiently using a different set of resources. Most of these applications, however, use a fixed resource topology even though certain tasks could benefit from using different resources. This can be due to the computational demands of these tasks or due to a change in resource availability over time. A wide range of work has been done on developing external management systems that allow applications to change grid resources during execution. This includes workflow systems [1,2,3] or grid schedulers with migration capabilities [4,5] that support resource switches that are either part of a predefined workflow or requested by the user.

An application management system that autonomously switches at run-time has been proposed by [6], where a hierarchically distributed application management system dynamically schedules and migrates a bag-of-tasks style MPI application, using a static hierarchy of schedulers to accomplish this.

A self-adaptive grid application that does not require external managers has been presented in [7]. Although this application does not use grid scheduling, it is able to autonomously migrate to different locations and change its number of processes. This has been accomplished by allowing all processes to share knowledge and cooperate in managing the application's topology.

In this work, we propose the living grid application, in which the application also decides where to run, and which is also able to migrate itself at run-time to another computer when needed. The intelligent migration from one computer to another can be realized over a long baseline, but does not need to be designed this way (see Sec. 2). We have applied this method to a multi-scale simulation on an intercontinental grid of semi-dedicated computers.

2 Living Application

2.1 Rationale

A flexible approach is needed to execute a complex grid application with multiple tasks and a diverse palette of resource requirements. The application should then be able to switch between tasks at run-time and between the resources required for each of these tasks, while maintaining the integrity of its data during these switches.

A switch requires the application to terminate its current execution, output its current state, and from that reinitialize the application using a new resource topology suited for the task at hand. Previously this has been done on a grid only in orchestration with a workflow manager. A job submitted by a workflow manager lacks the ability to change its resource topology during execution, as it does not have the privileges to make use of grid schedulers. When running an application with multiple tasks, this results in a 'bouncing' pattern where the manager submits jobs which return once a switch is required, only to be instantly submitted again to handle a different task. In the most favorable case, the performance loss introduced by bouncing and managerial overhead can be limited, but even then the successful completion of the simulation depends on the availability of an external manager, which is a potential single point of failure.

2.2 How the Living Application Works

The living application switches between sites and tasks dynamically and without external dependencies. It is based on four principles:

1. It makes decisions on which tasks to do and which resources to use.
2. It makes these decisions based on knowledge it has acquired at run-time.
3. It changes resources and switches between tasks.
4. It operates autonomously.

As a living application operates autonomously on the grid, it obtains its privileges on its own without interacting with an external workflow manager or user.

Upon initialization, the application is locally equipped with the tools and data to perform the required tasks and the criteria for switching between tasks or resource topologies. It is then submitted as a job to the grid with the initial resource requirements defined by the launcher. The living application begins execution on the grid and continues to do so until either a switch or a termination is required.

The conditions for switching or termination are determined prior to the start of the calculation or during run-time, but they are not necessarily static. They can rely on the internal state of the application, or on information from external sensors. When the conditions for a switch have been met, the application will migrate to different grid resources, switch to a different task, or both.

The switching between tasks requires two steps, which are finalizing the old task (and any program it still uses) and starting up the new task. During this switch, the application-specific data should be left intact. The switching between sites requires a larger number of actions, which are:

1. Creating a set of files consisting of the current application, files with its parameters and data and a script that specifies the methods and conditions for switching and termination.
2. Creating a job definition for the application on the new resources.
3. Authenticating (independently) on the grid.
4. Transferring the files to the remote site (if this is not done automatically by a resource broker).
5. Submitting the job, either through a resource broker or by directly accessing the head nodes of grid sites.
6. Reinitializing the living application on the new site.

Additional file transfer may be required, if the application has locally written data that is required elsewhere. The application could initiate the transfer of output files either during run-time (e.g. if separate files are written) or just before a job terminates on one machine (if data is appended to a single large file or data transfer would cause overhead at run-time).

The living application requires some user privileges to initiate data transfers and to autonomously migrate from one site to another. We obtain these privileges by using a grid client interface to access a credential management service. The details of this method are discussed in Sec. 2.3. The application requires access to the grid client interfaces on all participating nodes to request these privileges during execution. Once these privileges are granted, the application can perform authentication, data transfers and job submissions to the grid.

2.3 Security Considerations

User privileges on the grid are provided by an X.509 grid proxy [8] which requires the presence of a certificate, a private key and a correct pass phrase typed in by the user. This proxy is represented by a temporary file with limited lifetime. The easiest way to provide user privileges to a living application would be to equip it with this file, transporting it as it migrates, allowing it to reuse the proxy on remote locations. However, this approach has three drawbacks:

First, the presence of a proxy file on a remote site poses a security risk. If the file is not read-protected or stored in a shared account, it may be possible for other grid users to copy the proxy. The possession of this proxy enables them to impersonate the living application user for the duration of the proxy's lifetime, providing them with rights and resources that they could otherwise not use. Even if the proxy is on a dedicated account and read-protected, local users with admin rights are able to copy it and use it for impersonation.

Second, it is not possible to cancel the application after the first stage, as the proxy is initialized only at startup, after which it travels around on remote sites. This may cause a malfunctioning application to continue running and migrating until the proxy lifetime is exceeded. An application that is equipped for self-reproduction may iteratively spawns multiple successors which could lead to a grid meltdown.

Third, for the same reasons as before it is also not possible to prolong the lifetime of the proxy. This could cause the application to terminate prematurely once the proxy lifetime is exceeded. Specifying an excessively long lifetime relieves this problem, at the expense of increasing exposure to the other two drawbacks.

To reduce these drawbacks we have chosen to use an intermediary MyProxy server [9] in our implementation. The user initializes his or her proxy on the MyProxy server and defines a unique password. External sources use this password to access the server and initialize credentials with a limited life-time (normally set to 12 hours). The living application is provided with this password, and can therefore obtain these user privileges. If the password is stolen, others may be able to get the same privileges, but the user can block further access at any time by destroying the credential.

During application execution, the user can also extend the lifetime of his MyProxy credential by renewing it. It is also possible to replicate the credentials to other MyProxy servers, which allows the application to use remote MyProxy servers if the local server has died, rather than terminating itself upon switching.

2.4 Living Simulation

A special case of the living application is the living simulation. Today, simulations of complex systems, in which the dynamic range exceeds the standard precision of the computer, call for a wide range of numerical solvers [10]. Each of these solvers may run most efficiently on a different computer architecture. Most such simulations, however, are run on a single computer even though they would benefit from running on a variety of architectures.

This can be solved by migrating the application at run-time from one computer to another, in other words, by creating a living simulation. We demonstrate the concept of the living application by applying it to the (living) simulation of two galaxies merging.

The term living simulation has been previously defined as simulations that fine-tune their behavior at run-time based on input from external sensors, e.g. to provide input for performing adaptive load balancing [11]. In our definition we provide the simulation with user privileges and expect it to function autonomously.

3 Discussion

A living simulation is based on the principle that it autonomously switches between sites and solvers when required. This switching is done dynamically and without external dependencies. The simulation is locally equipped with the required solvers, the switching criteria and the initial conditions. It is then submitted as a job to the grid with the initial resource requirements defined by the launcher. The living simulation begins calculating on the grid and continues to do so until either a switching condition or a termination condition has been met.

By using the idea of the living applications, we have implemented and tested a living simulation, in which the merger of two galaxies, each with a central

Table 1. Specifications for the test nodes. The first column gives the name of the computer followed by its country of residence (NL for the Netherlands, US for the United States). The subsequent columns give the type of processor in the node, followed by the amount of RAM, the operating system, and the special hardware installed on the PC. Both nodes are connected to the internet with a 1 Gbit/s Ethernet card.

name	location	CPU type	RAM [MB]	OS	hardware
darkstar	NL	Core2Duo 3.0GHz	2048	Debian	Nvidia 8800 Ultra
zonker	US	2x Xeon 3.6GHz	2048	Gentoo	GRAPE 6A

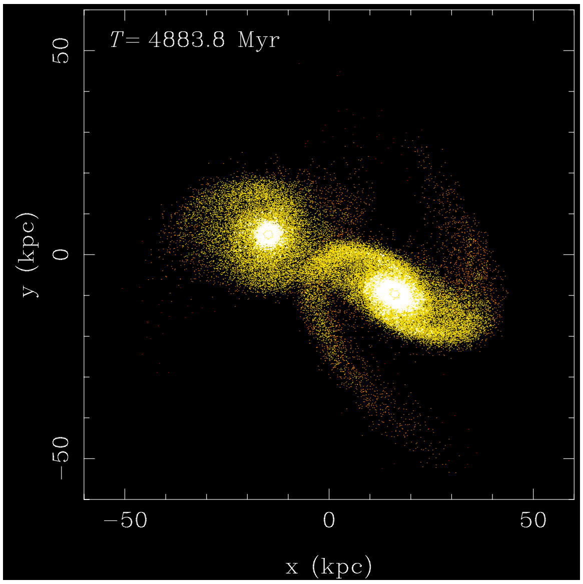


Fig. 1. Simulation snapshot of one of the runs, where the two galaxies approach for an initial interaction

supermassive black hole (SMBH), is simulated. We used a GPU-enabled tree code [12,13] for the early stages of the merger and switched to a GRAPE-enabled direct integrator [14] during later stages, when the separation between the two SMBHs was sufficiently small. We tested the overhead of our simulation using two machines with Globus middleware, one of which was equipped with GRAPE dedicated hardware (GRAvity PipE, [15]) and the other with a Nvidia 8800 GTX Ultra GPU. A specification for both nodes can be found in Tab. 1.

During our living simulation runs, the simulation performed three switches. The overhead caused by autonomously switching between machines was marginal, amounting to $\sim$ 4 percent of the three hours it for a 64k particle run to complete. A sample snapshot from on of our runs can be found in Fig. 1.

4 Conclusion

We introduced the living application as a way to manage complex applications on a large distributed infrastructure. Due to the autonomous nature of a living simulation, it is important to provide a mechanism that allows the user to terminate it. By having the simulation retrieve its extended privileges from a credential management service (MyProxy), users are able to revoke the privileges of the simulation regardless of its location. In addition, we can make repeated use of short-lived proxy credentials instead of a a single long-lived credential, which poses a larger security risk.

We have applied this concept in a living simulation of two galaxies merging. Our approach allows the simulation to use the optimal compute resources for each of the two solvers, switching resources whenever a different solver is required. In our example case, the solvers were a tree code and a direct N-body method, which were optimized for two kinds of special-purpose hardware, namely a GPU (tree) and a GRAPE (direct). The switches between these two solvers take place without user intervention, remote output retrieval or external managers. The execution time was only affected marginally by overhead such as caused by job migration and data transfer over the grid.

The creation of grid species enables us to give a simulation the ability to autonomously use the grid, acquire and apply internal knowledge, and migrate themselves. If we expand the awareness of a living simulation by letting it inherit more advanced sensing and scheduling abilities, we will be able to apply it to problems of greater complexity. In that way we could allow our simulation to evolve to a more complex organism.

Acknowledgements

We are grateful to Rick Quax for his contributions during the early stages of this project, and Steve McMillan for technical support and access to the GRAPE at Drexel University. We also thank David Groep and Hashim Mohammed for their input on the security aspects of this work, to Evghenii Gaburov for sharing his octgrav code for our experiments and to Alfons Hoekstra for his useful suggestions.

This research is supported by the Netherlands organization for Scientific research (NWO) grant #643.200.503, the European Commission grant for the QosCosGrid project (grant number: FP6-2005-IST-5 033883), the Netherlands Advanced School for Astronomy (NOVA) and the Leids Kerkhoven-Bosscha fonds (LKBF).

References

1. Herrera, J., Huedo, E., Montero, R., Llorente, I.: Porting of scientific applications to grid computing on gridway. Sci. Program. 13(4), 317–331 (2005)
2. Ludscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency and Computation: Practice and Experience 18(10), 1039–1065 (2006)
3. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. Journal of Grid Computing 3, 171–200 (2005)
4. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: A computation management agent for multi-institutional grids. Cluster Computing 5, 237–246 (2002)
5. Allen, G., Angulo, D., Foster, I., Lanfermann, G., Liu, C., Radke, T., Seidel, E., Shalf, J.: The cactus worm: Experiments with dynamic resource discovery and allocation in a grid environment. Int. J. High Perform. Comput. Appl. 15(4), 345–358 (2001)
6. Nascimento, A., Sena, A., Boeres, C., Rebello, V.: Distributed and dynamic self-scheduling of parallel mpi grid applications. Concurr. Comput.: Pract. Exper. 19(14), 1955–1974 (2007)
7. Wrzesinska, G., van Nieuwpoort, R., Maassen, J., Bal, H.: Fault-tolerance, malleability and migration for divide-and-conquer applications on the grid. In: IPDPS 2005: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005) - Papers, Washington, DC, USA. IEEE Computer Society, Los Alamitos (2005)
8. Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Gawor, J., Meder, S., Siebenlist, F.: X.509 proxy certificates for dynamic delegation. In: Proceedings of the 3rd Annual PKI R&D Workshop (2004)
9. Basney, J., Humphrey, M., Welch, V.: The myproxy online credential repository. Software: Practise and Experience 35(9), 801–816 (2005)
10. Hoekstra, A., Portegies Zwart, G., Bubak, S., Sloot, P.: Towards Distributed Petascale Computing. In: Bader, D.A. (ed.) Petascale Computing: Algorithms and Applications. Computational Science Series, 565 pp. Chapman & Hall/CRC, Boca Raton (2008)
11. Korkhov, V.V., Krzhizhanovskaya, V.V., Sloot, P.M.A.: A grid-based virtual reactor: Parallel performance and adaptive load balancing. J. Parallel Distrib. Comput. 68(5), 596–608 (2008)
12. Barnes, J., Hut, P.: A Hierarchical O(NlogN) Force-Calculation Algorithm. Nature 324, 446–449 (1986)

13. Gaburov, E., Nitadori, K., Harfst, S., Portegies Zwart, S., Makino, J.: A gravitational tree code on graphics processing units: Implementation in cuda (in preparation) (2009)
14. Harfst, S., Gualandris, A., Merritt, D., Spurzem, R., Portegies Zwart, S., Berczik, P.: Performance analysis of direct N-body algorithms on special-purpose supercomputers. New Astronomy 12, 357–377 (2007)
15. Sugimoto, D., Chikada, Y., Makino, J., Ito, T., Ebisuzaki, T., Umemura, M.: A Special-Purpose Computer for Gravitational Many-Body Problems. Nature 345, 33–35 (1990)

Applying Processes Rescheduling over Irregular BSP Application

Rodrigo da Rosa Righi[1], Laércio Lima Pilla[1], Alexandre Silva Carissimi[1], Philippe O.A. Navaux[1], and Hans-Ulrich Heiss[2]

[1] Instituto de Informática, Federal University of Rio Grande do Sul - Brazil
[2] Kommunikations- und Betriebssysteme, Technical University Berlin - Germany

Abstract. This paper shows an evaluation of processes rescheduling over an irregular BSP (Bulk Synchronous Parallel) application. Such application is based on dynamic programming and its irregularity is presented through the variation of computation density along the matrix' cells. We are using MigBSP model for processes rescheduling, which combines multiple metrics - Computation, Communication and Memory - to decide about processes migration. The main contribution of this paper includes the viability to use processes migration on irregular BSP applications. Instead to adjust the load of each process by hand, we presented that automatic processes rebalancing is an effortless technique to obtain performance. The results showed gains greater than 10% over our multi-cluster architecture. Moreover, an acceptable overhead from MigBSP was observed when no migrations happen during application execution.

1 Introduction

The dynamism and adaptivity are keywords in the new scenarios of parallel and distributed computing [1]. For example, the dynamism can be seen at the application level as well as a characteristic of the parallel machine architecture. In the first case, processes can change their amount of computation and/or their pattern of communication at any moment during application runtime. The dynamism related to infrastructure is demonstrated, for instance, through the fluctuation of processors load and availability, as well as in bandwidth due to either congestion in the network or modifications on its utilization. This kind of dynamism and uncertainty are challenges found in large distributed systems like grids [1]. Considering both ideas of dynamism, properties established previously may not be proven during runtime making dynamic scheduling pertinent. Dynamic scheduling can deal with the adaptivity issue since decisions managed during runtime can trigger changes in the scheduler behavior itself and adaptations that guarantee an acceptable level of application performance.

In this context, we address the issue of adaptivity through the control of processes migration over grid environments. We have developed a model called **MigBSP** that controls processes rescheduling on BSP (Bulk Synchronous Parallel) applications [2,3]. MigBSP performs an automatic load (processes) rebalancing among the resources without changing the application's code and acts without previous knowledge about the application's behavior. Our model can be seen as a dynamic rescheduler that treats both cases of dynamicity related above. Especially at application level, it uses two patterns - one for Computation and other for Communication - in order to measure the processes' regularity regarding their performed instructions as well as communicated bytes at each superstep. Besides computation and communication phases of a superstep, our model observes the costs to decide about the processes transferring. Finally, contrary to existing approaches [2], MigBSP performs the rescheduling according to the system state using an adaptable interval between calls for migration.

Aiming to perform an evaluation of MigBSP, we modeled an heterogeneous multi-cluster architecture and an irregular application. The application is based on dynamic programming (DP), which is a popular algorithm design technique for optimization problems [4]. In practice, there are many irregular DP applications where the workload varies across the matrix. Firstly, alternatives to achieve performance on such application include: (i) the mapping of the most loaded cells to faster processors; (ii) to determine appropriate data partitioning scheme on the fly or; (iii) to apply receiver-initiated load balancing algorithms inside the application. However, these techniques require an effort to recognize the target architecture previously and/or change application's code by hand. On the other hand, a possibility is to use automatic processes migration. In this way, our model acts at middleware level attempting to gain performance effortlessly.

This paper presents the impact of MigBSP over an irregular DP application. We will observe the impact of the load computation density on choosing processes for migration and on the regularity of rescheduling calls. As general ideas, migrations take place to faster resources (since the application is CPU-bound) and are considered viable as far as the load increases along the matrix.

2 MigBSP: Processes Rescheduling Model

Figure 1 (a) shows a superstep s in which the processes are not balanced among the resources. The main idea of MigBSP is to reduce the time of each superstep. Figure 1 (b) shows the expected result with processes redistribution after the end of superstep s. The architecture is heterogeneous and composed by clusters, supercomputers and local networks. The model requires that the involved nodes allow all-to-all asynchronous communications. The heterogeneous issue considers the processors' clock (all processors have the same architecture), as well as network speed and level (Fast and Gigabit Ethernet and multi-clusters environment, for instance). This architecture is assembled with Sets (different sites) and Set Managers. Set Managers are responsible for scheduling, capturing data from a specific Set and exchanging it among other managers.

MigBSP answers the following issues: (i) "When" to launch the migration; (ii) "Which" processes are candidates for migration; (iii) "Where" to put an elected process. In [3] we described the ideas to treat these questions in details. However, such work does not deal with irregular applications.

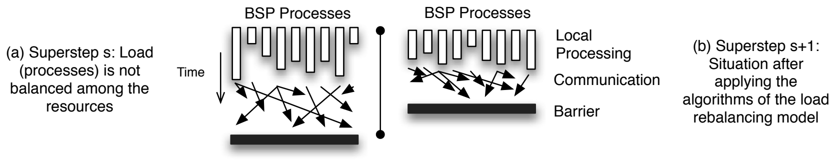

(a) Superstep s: Load (processes) is not balanced among the resources
Time
BSP Processes
BSP Processes
Local Processing
Communication
Barrier
(b) Superstep s+1: Situation after applying the algorithms of the load rebalancing model

Fig. 1. Supersteps in different situations

The decision for processes remapping is taken at the end of a superstep (after the barrier). This migration point was chosen because in this moment it is possible to analyze data from all BSP processes at their computation and communication phases. Aiming to generate the least intrusiveness in application as possible, we applied two adaptations that control the value of α. α is updated at each rescheduling call and will indicate the interval to the next one. Aiming to store the variations on system state, a temporary variable called α' is used and updated at each superstep through the increment or decrement of one unit. α is filled with α' value in the moment of rescheduling call. The adaptations' ideas are: (i) to postpone the rescheduling call if the system is stable (processes are balanced) or to turn it more frequent, otherwise; (ii) to delay this call if a pattern without migrations on ω past calls for rescheduling is observed. Aiming to analyze the system stability, the times of the slowest and fastest processes at each superstep are captured as well as the average time is computed. Only the processes that perform any computation activity enters to observe system stability. A variable D is used to indicate a percentage of how far the slowest and the fastest processes may be from the average to consider the processes balanced.

The answer for "Which" is solved through our decision function called Potential of Migration (PM). Each process i computes n functions $PM(i,j)$, where n is the number of Sets and j means a Set. The idea consists in performing a subset of the processes-resources tests at the rescheduling moment. $PM(i,j)$ is found using Computation, Communication and Memory metrics. The relation among them is based on the notion of force from physics. Computation and Communication act in favour of migration, while Memory works in an opposite direction. Computation metric - $Comp(i,j)$ - considers a Computation Pattern $P_{comp}(i)$ that measures the stability of a process i regarding the amount of instructions at each superstep. This value is close to 1 if the process is regular and close to 0 otherwise. This metric also performs a computation time prediction based on all computation phases between two activations of processes rescheduling. For this prediction it is used the Aging concept. In the same way, Communication metric - $Comm(i,j)$ - computes the Communication Pattern $P_{comm}(i,j)$ between processes and Sets. Furthermore, this metric uses communication time prediction considering data between two rebalancing activations. Memory metric - $Mem(i,j)$ - considers process memory, transferring rate between considered process and the manager of target Set, as well as migration costs.

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \qquad (1)$$

$PM(i,j)$ selects the candidate processes for migration (see Equation 1). A high $PM(i,j)$ means that process i has high computation time, high communication with processes that belong to Set j and presents low migration cost. BSP processes calculate $PM(i,j)$ locally. At each rescheduling call, each process passes its highest $PM(i,j)$ to its Set Manager. This last entity exchanges the PM of its processes among other managers. There are two heuristics to choose the candidates, both based on a decreasing ordered list of PMs. The heuristics are: (i) processes that have PM higher than a $MAX(PM) \times x$ are candidates, where x is a percentage; (ii) choose just one process.

$PM(i,j)$ of a candidate process i is associated to a Set j. The manager of this Set will select the most suitable processor to receive the process i. Before any migration, its viability is verified considering the following data: (i) the external load on source and destination processors; (ii) the BSP processes that both processors are executing; (iii) the simulation of considered process running on destination processor; (iv) the time of communication actions considering local and destination processors; (v) migration costs. Concerning this, we computed two times: $t1$ and $t2$. $t1$ means the local execution of process i, while $t2$ encompasses its execution on the other processor and includes the migration costs. For each candidate is chosen a new resource (if $t1 > t2$) or its migration is canceled.

3 Irregular DP Application

Dynamic programming is a method of solving problems exhibiting the properties of overlapping subproblems and optimal substructure [4]. DP algorithms can be classified according to the matrix size and the dependency relationship of each matrix cell. An algorithm for a problem of size n is called tD/eD if its matrix size is $O(n^t)$ and each matrix cell depends on $O(n^e)$ other cells. In this paper we work with 2D/1D DP algorithms, which are all irregular with load changes along the matrix's cells. In particular, we observed the functioning of Smith-Waterman algorithm [5] that is a well-known algorithm for local sequence alignment.

Smith-Waterman algorithm proceeds in a series of wavefronts diagonally across the matrix. Figure 2 (a) illustrates a 4×4 matrix with a column-based processes allocation. The more intense the shading, the greater is the load computation density of the cell. Each wavefront corresponds to a superstep. For instance, Figure 2 (b) shows a 4×4 matrix with 7 supersteps. This organization

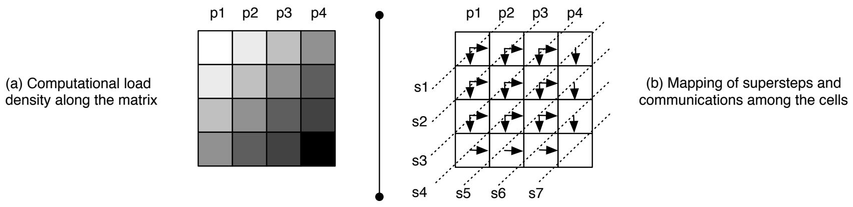

(a) Computational load density along the matrix
p1
p2
p3
p4
(b) Mapping of supersteps and communications among the cells
p1
p2
p3
p4
s1
s2
s3
s4
s5
s6
s7

Fig. 2. Irregular BSP dynamic application organization

brings the following conclusions: (i) $2n - 1$ supersteps are computed in a square matrix with order n and; (ii) each process will be involved on n supersteps.

At each superstep, each process computes a block of data and sends it to other process. Figure 2 (b) shows the communication among the processes. Considering that cell a, b (a means a matrix' line, while b is a matrix' column) needs data from the $a, b - 1$ and $a - 1, b$ other ones, we have an interaction from process pb to $pb + 1$. We do not have communication inside the same column.

4 Evaluation Methodology

We are using simulation in three scenarios: (i) Application execution simply; (ii) Application execution with scheduler without applying migrations; (iii) Application execution with scheduler allowing migrations. Scenario ii indicates the overhead associated with scheduling calculus and message passing between processes and Set Managers, as well as among Set Managers properly. The analysis of scenarios i and iii shows the performance when migrations are applied.

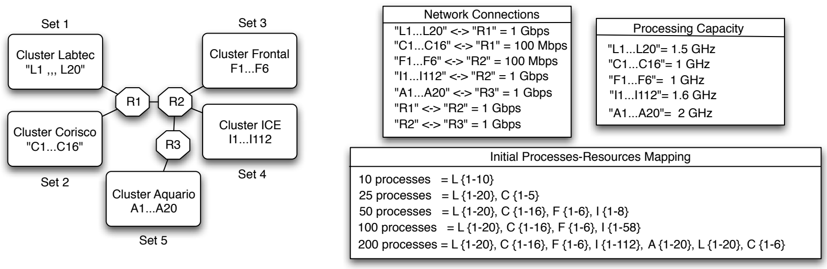
The configuration of scenarios ii and iii depends on the Computation Pattern $P_{comp}(i)$ of each process i. $P_{comp}(i)$ increases or decreases depending on the prediction of the amount of performed instructions at each superstep. $PI_t(i)$ represents this prediction for superstep t and process i. It is based on the Aging concept, in which uses the idea that the prediction is more strongly influenced by recent values. The formula to compute $PI_t(i)$ is shown below. k means superstep 1 or the superstep after the last call for processes rescheduling. $I_t(i)$ represents the amount of instructions executed by process i during superstep t.

$$PI_t(i) = \begin{cases} I_t(i) & if\ t = k \\ \frac{1}{2}PI_{t-1}(i) + \frac{1}{2}I_t(i) & if\ k < t \leq k + \alpha - 1 \end{cases}$$

$P_{comp}(i)$ is updated at each superstep following the Algorithm 1. We consider a specific process as regular if the forecast is within a δ margin of fluctuation from the amount of instructions performed. In our experiments, we are using 10^6 as the amount of instructions for the first superstep and 10^9 for the last one. The increase of load computational density among the supersteps is uniform. Considering this, we applied δ equal to 0.01 (1%) and 0.50 (50%) to scenarios ii and iii, respectively. This last value was used because $I_2(1)$ is 565.10^5 and $PI_2(1)$ is 287.10^5 when a 10×10 matrix is tested (19 supersteps). The percentage of 50% enforces instruction regularity in the system. Both values of δ will influence the Computation metric, and consequently the choosing of candidates for migration. Scenario ii tends to obtain negatives values for PM since the Computation Metric will be close to 0. Consequently, no migrations will happen on this scenario. Following with the parameters, we tested the behavior of square matrixes of order 10, 25, 50, 100 and 200. In addition, we performed a column-based mapping, where process pi is responsible for column i of the matrix. Each cell of a 10×10 matrix needs to send 500 Kbytes and each process occupies 1.2 Mbyte in memory (700 Kbytes for other data). The cell of 25×25 matrix communicates 200 Kbytes and each process occupies 900 Kbytes in memory and so on.

Algorithm 1. Computation Pattern $P_{comp}(i)$ of the process i

```
1: for $t$ from superstep $k$ to superstep $k + \alpha - 1$ do
2:     if $PI_t(i) \geq I_t(i).(1 - \delta)$ and $PI_t(i) \leq I_t(i).(1 + \delta)$ then
3:         Increases $P_{comp}(i)$ by $\frac{1}{\alpha}$ up to 1
4:     else
5:         Decreases $P_{comp}(i)$ by $\frac{1}{\alpha}$ down to 0
6:     end if
7: end for
```



Network Connections
"L1...L20" <-> "R1" = 1 Gbps
"C1...C16" <-> "R1" = 100 Mbps
"F1...F6" <-> "R2" = 100 Mbps
"I1...I112" <-> "R2" = 1 Gbps
"A1...A20" <-> "R3" = 1 Gbps
"R1" <-> "R2" = 1 Gbps
"R2" <-> "R3" = 1 Gbps

Processing Capacity
"L1...L20"= 1.5 GHz
"C1...C16"= 1 GHz
"F1...F6"= 1 GHz
"I1...I112"= 1.6 GHz
"A1...A20"= 2 GHz

Initial Processes-Resources Mapping	
10 processes	= L {1-10}
25 processes	= L {1-20}, C {1-5}
50 processes	= L {1-20}, C {1-16}, F {1-6}, I {1-8}
100 processes	= L {1-20}, C {1-16}, F {1-6}, I {1-58}
200 processes	= L {1-20}, C {1-16}, F {1-6}, I {1-112}, A {1-20}, L {1-20}, C {1-6}

Fig. 3. Testbed infrastructure and the initial processes-resources mappings

We are using the Simgrid [6] (MSG module). This simulator is deterministic, where a specific input always results in the same output. We assembled an infrastructure with five Sets, as we can see in Figure 3. Each node has a single processor. These Sets represent a real infrastructure with five clusters located at Federal University of Rio Grande do Sul, Brazil. For sake of simplicity, we hide the network of each cluster. Clusters Labtec, Corisco and Frontal have their nodes linked by Fast Ethernet, while ICE and Aquario use Gigabit connection. The migration costs are based on executions with AMPI [7] on our clusters.

Figure 3 also presents the initial processes-recourses mappings. Finally, initial tests were executed using α equal to 2, 4, 8 and 16. Furthermore, we employed ω equal to 3, initial D equal to 0.5 and used heuristic one to choose the candidates for migration with x equal to 80%. We observed that our testbed environment prioritized the heterogeneity issue. Future works include the execution of BSP applications and MigBSP over dynamic environments. Simgrid allows to write files informing the variation in time of bandwidth, latency and CPU capacities.

5 Evaluation and Discussions

Table 1 presents the application evaluation. 19 supersteps were crossed when a 10×10 matrix was tested. Adopting this size of matrix and α 2, 13.34s and 14.15s were obtained for scenarios i and ii which represents a cost of 8%. The higher is the value of α, the lower is the MigBSP overhead on application execution. This occurs because the system is stable (processes are balanced) and α always increases at each rescheduling call. Three calls for processes relocation were done when testing α 2 (at supersteps 2, 6 and 14). The rescheduling call at superstep 2 does not produce migrations. At this step, the load computational density is not enough to overlap the consider migration costs involved on process transferring operation. The same occurred on the next call at superstep 6. The last call happened at superstep 14, which resulted on 6 migrations: {(p5,a1), (p6,a2), (p7,a3), (p8,a4), (p9,a5), (p10,a6)}. MigBSP indicated the migration of processes that are responsible to compute the final supersteps. The execution with α equal to 4 implies in a shorter overhead since two calls were done (at supersteps 4 and 12). Observing scenario iii, we do not have migrations in the first call, but eight occurred in the other one. Processes 3 up to 10 migrated in this last call to cluster Aquario. α 4 outperforms α 2 for two reasons: (i) it does lesser rescheduling calls and; (ii) the call that causes processes migration was done at a specific superstep in which MigBSP takes better decisions.

Table 1. Evaluation of scenarios i, ii and iii when varying the matrix size

Scenarios		mat. 10×10	mat. 25×25	mat. 50×50	mat. 100×100	mat. 200×200
Scenario i		13.34s	40.74s	92.59s	162.66s	389.91s
Scen. ii	$\alpha = 2$	14.15s	43.05s	95.70s	166.57s	394.68s
	$\alpha = 4$	14.71s	42.24s	94.84s	165.66s	393.75s
	$\alpha = 8$	13.78s	41.63s	94.03s	164.80s	392.85s
	$\alpha = 16$	13.42s	41.28s	93.36s	164.04s	392.01s
Scen. iii	$\alpha = 2$	13.09s	35.97s	85.95s	150.57	374.62s
	$\alpha = 4$	11.94s	34.82s	84.65s	148.89s	375.53s
	$\alpha = 8$	13.82s	41.64s	83.00s	146.55s	374.38s
	$\alpha = 16$	12.40s	40.64s	85.21s	162.49s	374.40s

The system stays stable when the 25×25 matrix was tested. α 2 produces a gain of 11% in performance when considering 25×25 matrix and scenario iii. This configuration presents four calls for processes rescheduling, where two of them produce migrations. No migrations are indicated at supersteps 2 and 6. Nevertheless, processes 1 up to 12 are migrated at superstep 14 while processes 21 up to 25 are transferred at superstep 30. These transferring operations occurred to the fastest cluster. In this last call, the remaining execution presents 19 supersteps (from 31 to 49) to amortize the migration costs and to get better performance. The execution when considering α 8 and scenario iii brings an overhead if compared with scenario i. Two calls for migrations were done, at supersteps 8 and 24. The first call causes the migration of just one process (number 1) to a1 and the second one produces three migrations: {(p21,a2),(p22,a3),(p23,a4)}. We observed that processes p24 and p25 stayed on cluster Corisco. Despite performed migrations, these two processes compromise the supersteps that include them. Both are executing on a slower cluster and the barrier synchronization always waits for the slowest process. Maintaining the matrix size and adopting α 16, we have two calls for migration: at supersteps 16 and 48. This last call migrates p24 an p25 to cluster Aquario. Although this movement is pertinent to get performance, just one superstep is executed before finalizing the application.

50 processes were evaluated when the 50×50 matrix was considered. In this context, α also increases at each call for processes rescheduling. We observed that an overhead of 3% was found when scenario i and ii were compared (using α 2). In addition, we observed that all values of α achieved a gain of performance in scenario iii. Especially when α 2 was used, five calls for processes rescheduling were done (at supersteps 2, 6, 14, 30 and 62). No migrations are indicated in the first three calls. The greater is the matrix size, the greater is the amount of supersteps needed to make migrations viable. This happens because our total load is fixed (independent of the matrix size) but the load partition increases uniformly along the supersteps (see Section 4 for details). Process 21 up to 29 are migrated to cluster Aquario at superstep 30, while process 37 up to 42 are migrated to this cluster at superstep 62. Using α equal to 4, 84.65s were obtained for scenario iii which results a gain of 9%. This gain is greater than that achieved with α 2 because now the last rescheduling call is done at superstep 60. The same processes were migrated at this point. However, there are two more supersteps to execute using α equal to 4. Three rescheduling calls were done with $\alpha 8$ (at supersteps 8, 24 and 56). Only the last two produce migration. Three processes are migrated at superstep 24: {(p21,a1),(p22,a2),(p23,a3)}. Process 37 up to 42 are migrated to cluster Aquario at superstep 56. This last call is efficient since it transfers all processes from cluster Frontal to Aquario.

The execution with a 100×100 matrix shows good results with processes migration. Six rescheduling calls were done when using α 2. Migrations did not occur at the first three supersteps (2, 6 and 14). Process 21 up to 29 are migrated to cluster Aquario after superstep 30. In addition, process 37 to 42 are migrated to cluster Aquario at superstep 62. Finally, superstep 126 indicates 7 migrations, but just 5 occurred: p30 up to p36 to cluster Aquario. These migrations complete one process per node on cluster Aquario. MigBSP selected for migration those processes that belonged to cluster Corisco and Frontal, which are the slowest clusters on our infrastructure testbed. α equal to 16 produced 3 attempts for migration when a 100×100 matrix is evaluated (at supersteps 16, 48 and 112). All of them triggered migrations. In the first call, the 11^{th} first processes are migrated to cluster Aquario. All process from cluster Frontal are migrated to Aquario at superstep 48. Finally, 15 processes are selected as candidates for migration after crossing 112 supersteps. They are: p21 to p36. This spectrum of candidates is equal to the processes that are running on Frontal. Considering this, only 3 processes were migrated actually: {(p34,a18),(p35a19),(p36,a20)}.

Table 1 also shows the application performance when the 200×200 matrix was tested. The overhead of our model is smaller than 2% for scenario ii. Furthermore, satisfactory results were obtained with processes migration. The system stays stable during all application execution. Despite having more than one process mapped to one processor, sometimes just a portion of them is responsible for computation at a specific moment. This occurs because the processes are mapped to matrix columns, while supersteps comprise the anti-diagonals of the matrix. Using α 2 and considering scenario iii, 8 calls for processes rescheduling were done. Migrations were not done at supersteps 2, 6 and 14. Processes 21 up to 31 are migrated to cluster Aquario at superstep 30. Moreover, all processes from cluster Frontal are migrated to Aquario at superstep 62. Six processes are candidates for migration at superstep 126: p30 to p36. However, only p31 up to p36 are migrated to cluster Aquario. These migrations happen because the processes initially mapped to cluster Aquario do not collaborate yet with BSP computation. Migrations are not viable at superstep 254. Finally, 12 processes (p189 to p200) are migrated to cluster Aquario when superstep 388 was crossed. At this time, all previous processes allocated to Aquario are inactive and the migrations are viable. However, just 10 remaining supersteps are executed to amortize the processes migration costs.

6 Related Work

GridWay treats with time and cost optimization scheduling and migration [8]. Both migration mechanisms consider only data from CPU, like speed and load. Bhandarkar, Brunner and Kale presented a support for adaptive load balancing in MPI applications [9]. Periodically, the application transfers control to the load balancer using a special call. These authors present a Metis-based strategy that uses the communication graph to remap the processes. AMPI [7] uses Charm++ to offer automatic load balancing. Charm++ collects workload data and the objects communication pattern. At each load balancing time, the load balancer uses such data to redistribute the workload. Vadhiyar and Dongarra presented a migration framework and self adaptivity in GrADS [1]. However, they computed the migration cost as a fixed value. In addition, the gain with rescheduling is based on the remaining execution time prediction over a new resource. Thus, this work must deal with applications in which their parts are known in advance.

Concerning the BSP scope, Jiang, Tong and Zhao presented resource load balancing based on multi-agents in ServiceBSP [10]. Load balancing is launched when a new task is inserted and is based on the load rank of the nodes. Scheduling service sends a new task to the current lightest node. Load value is calculated taking such information: CPU, memory, number of current tasks, response time and number of network links. In addition, we can cite two works that present migration on BSP applications. The first one describes the PUBWCL library, which aims to take profit of idle cycles from nodes around the Internet [2]. PUBWCL offers migration at each superstep. All proposed algorithms just use data about the nodes and consider the computation times from each process. Other work comprises the PUB library [11]. The author explains that a load balancer decides when to launch the migration, but this issue is not addressed in [11]. He proposed centralized and a distributed strategies for load balancing. In distributed approach, every node chooses c other nodes randomly and asks them for their load. One process is migrated if the minimum load of the c analyzed nodes is smaller than the load of the node that is performing the test. The drawback of this strategy is that it can create a lot of scheduling messages.

7 Conclusion and Future Works

This paper presented MigBSP briefly, as well as the treatment of an irregular application executed using it. MigBSP combines data from multiple metrics to create our decision function PM. PM considers the migration of a process i to a Set j. Thus, we do not test all possibilities at the rescheduling moment. The contribution of this paper is the possibility to use MigBSP over heterogeneous environments in order to get performance on irregular BSP applications. Initially, MigBSP was developed to deal with processes that present a regular behavior. However, it allows to fill parameters that turn possible to adjust its functioning to treat irregularity in an efficient manner. Section 5 presented the results and a discussion about the application execution. The main conclusions are: (i) a simple way to obtain performance is designing α in order to trigger the rescheduling call close to the end of the application, since MigBSP tends to select those processes that have more load; (ii) the greater is the load computational density of the last supersteps, the better will be the results with the migration of the last processes and; (iii) the application behavior implies that the processors can vary their load during the crossing of supersteps, changing their viability to receive processes.

Future works include simulation of MigBSP over the Grid5000. We intend to test MigBSP's scalability over this platform.

References

1. Vadhiyar, S.S., Dongarra, J.J.: Self adaptivity in grid computing: Research articles. Concurr. Comput.: Pract. Exper. 17(2-4), 235–257 (2005)
2. Bonorden, O., Gehweiler, J., auf der Heide, F.M.: Load balancing strategies in a web computing environment. In: Conf. on Parallel Processing and Applied Mathematics, pp. 839–846 (2005)
3. Righi, R., Pilla, L., Carissimi, A., Navaux, P.O.A.: Controlling processes reassignment in bsp applications. In: 20th International Symposium on Computer Architecture and high Performance Computing, pp. 37–44. IEEE, Los Alamitos (2008)
4. Low, M.Y.H., Liu, W., Schmidt, B.: A parallel bsp algorithm for irregular dynamic programming. In: Xu, M., Zhan, Y.-W., Cao, J., Liu, Y. (eds.) APPT 2007. LNCS, vol. 4847, pp. 151–160. Springer, Heidelberg (2007)
5. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
6. Casanova, H., Legrand, A., Quinson, M.: Simgrid: A generic framework for large-scale distributed experiments. In: Conf. Comp. Modeling and Simulation, pp. 126–131. IEEE, Los Alamitos (2008)
7. Huang, C., Zheng, G., Kale, L., Kumar, S.: Performance evaluation of adaptive mpi. In: Symp. on Principles and pract. of parallel programming, pp. 12–21. ACM Press, New York (2006)
8. Moreno-Vozmediano, R., Alonso-Conde, A.B.: Influence of grid economic factors on scheduling and migration. In: Daydé, M., Dongarra, J., Hernández, V., Palma, J.M.L.M. (eds.) VECPAR 2004. LNCS, vol. 3402, pp. 274–287. Springer, Heidelberg (2005)

9. Bhandarkar, M.A., Brunner, R., Kale, L.V.: Run-time support for adaptive load balancing. In: IPDPS 2000: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, pp. 1152–1159. Springer, London (2000)
10. Jiang, Y., Tong, W., Zhao, W.: Resource load balancing based on multi-agent in servicebsp model. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007 (3). LNCS, vol. 4489, pp. 42–49. Springer, Heidelberg (2007)
11. Bonorden, O.: Load balancing in the bulk-synchronous-parallel setting using process migrations. In: Symp. on Parallel and Distrib. Processing, pp. 1–9. IEEE, Los Alamitos (2007)

Software Services and Tools

Support for Urgent Computing Based on Resource Virtualization*

Andrés Cencerrado, Miquel Ángel Senar, and Ana Cortés

Departamento de Arquitectura de Computadores y Sistemas Operativos
08193 Bellaterra (Barcelona), Spain
andres.cencerrado@caos.uab.es
{miquelangel.senar,ana.cortes}@uab.es

Abstract. Virtualization technologies provide flexible execution environments that could bring important benefits for computational problems with strong deadlines. Large Grid infrastructures are becoming available nowadays and they could be a suitable environment to run such on-demand computations that might be used in decision-making processes. For these computation, we encounter the need to deliver as much resources as possible at particular times. These resources may be provided by different institutions belonging to a grid infrastructure but there are two important issues that must be satisfied. Firstly, all resources must be correctly configured and all the components needed by the application must be properly installed. If there is something small missing that is required then applications will fail. Secondly, the execution of urgent applications must be made quickly in order to produce useful results in time. If applications must wait in a queue, results might be useless because they are obtained too late. To address these issues, we describe a job management service, based on virtualization techniques, that avoids configuration problems and increases the number of available resources to run applications with critical deadlines. We describe the main components of our service that can be used on top of common batch queue systems and we show some experimental results that prove the benefits of applying time-sharing techniques on the virtual machines to increase the performance of urgent computations.

1 Introduction

Nowadays, scientific community rely on computational resources in order to solve most of the present scientific problems. In order to satisfy computing-intensive problems in the minimum possible time, since the beginning of the 90s it is possible to resort to distributed computing, because of the deployment of grid infrastructures. These kind of infrastructures allow the users to count on many computing resources managed in a decentralized way.

Besides, there is an emerging area in the scientific applications field, Urgent High Performance Computing (Urgent HPC), which requires a great capacity

* This work is supported by the MEC-Spain under contracts TIN 2007-64974.

of computing resources at a given time. Urgent HPC applications could take benefit from a grid environment prepared to respond efficiently, because of the great resources offer and variety it can supply. In fact, there are several projects in progress which main goal is to get response from a multiple-resource sites as quick as possible [1], but due to the grid environments nature, it is necessary to propose new management strategies that provide not only a quick response, but a more efficient use of the available resources for the results to be more accurate in a shorter period of time.

Grid computing can be considered as a consolidated field in high performance computing, however, it still presents serious limitations from the point of view of Urgent HPC requests. Response time is an usual handicap in such environments. Each administrative domain, in a grid infrastructure, has its own components that take care about security and scheduling issues. All these components introduce considerable delay to the jobs starting, which is a clear penalty for urgent computing applications.

Nevertheless, time response is not the only constraint, compatibility between the application to be launched and the execution platform is a very important handicap as well. Libraries, permissions and the host operating system are examples of factors that can compromise the compatibility and, in consequence, the successful execution of the jobs. Many times, users are not aware of the requirements of their own jobs. This kind of *ignorance* usually leads to a considerable waste of time trying to guarantee the correct execution of the application on the remote sites and sometimes it turns out to be an impossible task.

Moreover, most of scientific applications rely on models/simulators that do not provide accurate results and the quality of the results will depend on how many times the underlying simulation can be executed before a deadline. Therefore, in those cases, as much executions we can deliver at a given time, the better the results become. Consequently, we focus on increasing the number of suitable working nodes for the underlying urgent application at a given time independently on the sites' particular configuration.

Virtual Machines (VMs) are becoming popular in grid computing as they provide a way to abstract the grid resources and allow grid applications to be run without worrying about the underlying platform of the resource. In this work, we propose a job-management service based on virtualization techniques, that avoid configuration problems and increases the number of available resources to run applications with critical deadlines.

This paper is organized as follows. In the next section, the main features of virtual machines are described. In section 3, we describe the architecture of the job management architecture. The experimental study is reported in section 4 and, finally, the main conclusions are included in section 5.

2 Virtual Machines for Urgent Computing

As we have previously mentioned, virtual machines provide an abstraction of the underlying computing resources allowing any kind of application to be executed without misleading problems. In particular, VMs provide several important advantages:

a) **Physical platform isolation:** thanks to the abstraction carried out by the virtual machine monitor (VMM), tasks execution does not depend on the host device features. Thus, any VM instance on which user can execute his application successfully will be useful and, therefore, any working node (WN) in the site can satisfy application demands.
b) **Multiplexation capability:** depending on the features of the host platform, it can support multiple VM instances execution, which could provide better performance per WN.
c) **Possibility of deploying new subpool of WNs:** a VM, like a real physical machine, can execute the software needed to become a new WN into the site, able to receive and process different tasks as well as physical WNs do. Furthermore, the fact of having multiple VMs as a part of the working nodes set of the site, allow us to deploy a new subset, or subpool, of virtual WNs. The idea besides that is the one referred as to *VM recycling*. Instead of starting and stopping a VM each time the submitted job requires this kind of execution environment, *VM recycling* deals with the idea of not stopping the VM and keeping it *alive* waiting for new tasks to be executed. This recycling strategy saves time, as the time needed to instantiate the VM is skipped.

As one can see, these features solve, or at least relieve, the main constraints of grid environments when dealing with an urgent HPC application. How each one of the above mentioned handicaps are overcome is subsequently listed.

– Compatibility constraints elimination: in order to be able to execute the application in any grid site, it is sufficient to have a VM that emulates a trusted system on which the desired application executes successfully.
– Response time reduction: as it has been previously mentioned, *VM recycling* permit to deploy subpools of virtual WNs that could be managed in a different way than the physical ones, establishing, for example, different priority polices, even up to dedicating it exclusively for particular applications.
– Turnaround time reduction: because of the multiplexation capability, it is possible to carry out more executions per time unit, reducing the turnaround time of each individual task/application execution compared to a standard batch execution.

As stated previously, this work is focused on efficient managing strategies of the grid-available resources. To achieve this goal, we propose a grid site local-level architecture design based on the use of Virtual Machines (VMs), as a suitable platform for grid infrastructures in order to allow an efficient and successful processing of urgent scientific applications under such environments.

Although this approach seems very promising, it also raises new difficulties and new challenges that must be explored, for instance, it is needed an accurate analysis of possible performance degradations as well as collateral effects. These aspects will be studied in the subsequent sections.

3 Architecture Design

Virtual machines in grid computing can be implemented in different ways depending on the purpose and intention of the grid. We shall now briefly describe two of the most significant approaches according to our work:

1.-Virtual Machines as Grid Infrastructure: this method allows Grids to be set up and deployed easily. Grid middleware is installed and configured within virtual machines, and users can create and use a Grid infrastructure by installing and deploying these virtual machine images. Machines are added and removed from the Grid when users start and terminate their virtual machines. One example of this virtual machine grid computing implementation is Grid Appliance [2].

2.- Virtual Machines running on Grid Infrastructures: this method is based on the use of an already existing infrastructure for supporting grid computing. Virtual Machines are executed as processes using the grid middleware just like normal applications. Some examples are In-VIGO [3] and Maestro-VC [4].

Our proposal fits the second approach, taking into account the possibility of dealing with virtual resources as new WNs inside the site, as well. Figure 1 depicts a modular schema of the proposed architecture for an urgent job execution site based on virtual resources. In this approach, Grid middleware layer is based on EGEE gLite framework [5]. As one can see, there are new resource components as VMs which have been deployed as a consequence of the execution of an ordinary task in a physical node, consisting of VM instance starting. These virtual nodes have the capability to become new WNs into the site. Our contribution, by means of this design, consists of the *Virtual Spaces Job Manager* (VSJM) component description, which is the responsible for the appropriate deployment and management of these virtual resources.

VSJM manages the virtual resources subpool, establishing direct connection with it, in order to be able to diversify efficiently incoming virtual-resource demanding jobs. This component is constituted by three subcomponents:

- Execution manager: responsible for carrying out scheduling tasks for this sort of jobs.
- VM manager: the subcomponent in charge of VM instantiation, by submitting a VM starting up job to the Local Resource Management System (LRMS), stopping, recycling and even migrating it.
- Job manager: this component will deal with the submission and control of tasks to be executed in virtual resources.

In order to include to the system the capability to correctly attend incoming requests, we lean on an extension of the EU-Datagrid Job Description Language [6], that is, the users, by means of a JDL file, may specify which virtual environment should be provided for their jobs. Figure 2 shows a simplified example of an extended JDL file where we include two new attributes: the *VMId*, which univocally specifies what VM image should be deployed, and the *UrgencyLevel*,

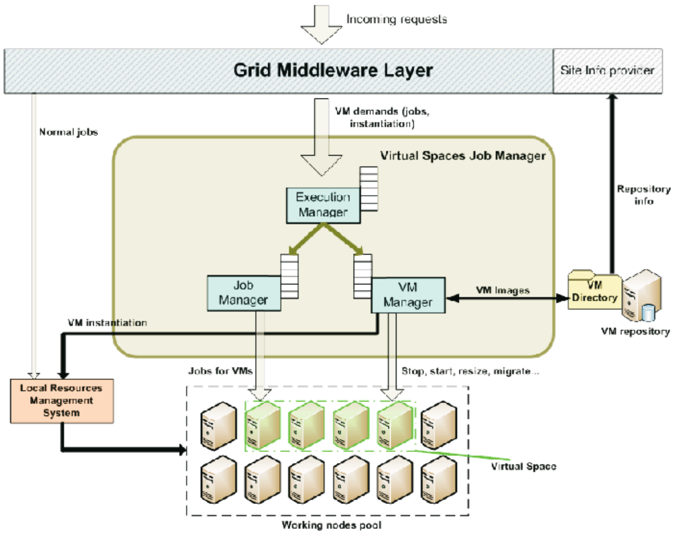


Fig. 1. Virtual resources-based site architectural design approach

```
Executable                  = "fusion_app";
Arguments                   = "-n";
JobType                     = "Normal";
InputSandBox                = {"fusion_app"};
VMId                        = e97315e469fb;
UrgencyLevel                = 1;
```

Fig. 2. Extended JDL job description

which specifies, in a 0 to 2 scale, the emergency degree of the incoming job, in order to efficiently attend this kind of jobs.

Based on this extended JDL file, users can order the instantiation of a VM image specifying its VMId, which had been established once the image had been deposited on the *VM repository*. Our objective is to adapt this component to the Globus Virtual Workspaces virtual repositories management system [7] [8].

Thus, jobs get into the system through the Grid middleware layer, which determines whether jobs will be executed in a physical WN, steering them directly to the LRMS, or jobs will be executed in a virtual WN, steering them in this case to the VSJM module. Additionally, jobs that demand a virtual environment, which is not already deployed, will cause the VM manager to act submitting an *ordinary* job to the LRMS consisting of VM starting up.

This architectural approach constitutes a solution that provides useful services for urgent computing, from which outstand the following ones:

– **Allowing usual job execution:** new functions added by our system do not interfere in the way the site operated previously.

- **Allowing VM instances executions (offered by the site or supplied by the user):** our approach contemplates the possibility of offering a series of VM images available in the site (stored in a storage element), and accepting user images as well, depending on authorization policies.
- **VMs capability to become WNs and constituting a new WN subset:** our system submits VMs images as normal jobs to the Local Resource Management System without requiring any special feature.
- **Transparent job submitting to VM instances:** our system includes non-intrusive techniques that allow the automatic instantiation of a resource management system (such as Condor [9]) when a VM image is started. This resource management system does not interfere with the Local Resource Management System deployed in the site and is used only by our Job Manager in order to submit jobs directly to VM instances. By exploiting disk images inclusion techniques, there is no need to modify the users' VM image nor to make them install any extra software in their images.
- **Efficient VMs recycling management:** the automatic deployment of a resource management system when each VM is started allows the execution of multiple jobs in the same VM instance. Therefore, significant saving of time is obtained by avoiding the overhead incurred in VM instantiation.
- **Basic infrastructure that can be able to dynamically increase the number of resources in case of emergency:** our system enables the development of intelligent scheduling strategies that can determine in an automatic and dynamic way when it is necessary to deploy new VM instances in order to attend to the incoming tasks load, without the need for the user to send VMs execution jobs.

4 Performance Analysis and Experimentation

This section describes some experimental results obtained with a first prototype that supports the basic functionality of the job management service described in the previous section. The goal is to demonstrate feasibility of the design and show the potential use of the virtual machine multiprogramming mechanism. Our prototype is interfaced to Condor as Local Resource Management System. The current stable version of Condor (7.0.4) provides direct support for managing VM images (which includes transferring images from a submission machine to an execution machine, starting and stopping the virtual machine, and so on). Condor was running on a testbed made of 12 Pentium IV machines, running Linux. The Grid middleware used for accessing the testbed was based on gLite. And we extended the existing gLite's job manager with additional services that provide a basic control of virtual images and application jobs.

We carried out a set of experiments with this prototype to measure the performance of two applications running in a virtualized mode and the effect of virtual machine multiprogramming on application performance. Our multiprogramming scheme allows the execution of two or more virtual machines on a physical machine. Each virtual machine can be used for a different application thus allowing urgent jobs to time share the CPU with other jobs running on a different virtual machine. A graceful degradation in performance is observed but this scheme does not require to kill or checkpoint a running job when an urgent job enters into the system.

We have used two computing-intensive applications to carry out the experimental analysis presented in this section. The first one is a *Dynamic Data Driven Genetic Algorithm* for fire spread prediction (DDDGA) [10]. The second one is BLAST [11], a well known biology application, based on searching regions of local similarity between genetic sequences. Both applications are compute-intensive. However, in our experiments we have used DDDGA as an example of an urgent job that was time shared with a non-urgent job (BLAST).

There is no appreciable cost (in terms of execution time and overhead) derived from the use of VMs as computational tool instead of a physical machine in the case of compute-intensive applications [12]. Table 1 shows execution times of both applications under VM and under native host.

Table 1. DDDGA and BLAST execution times (minutes)

Native host		Virtual Machine	
BLAST	DDDGA	BLAST	DDDGA
58.5	10	62	11

We carried out another experiment to demonstrate the fact that it is possible to implement an adaptive scheduling system which is able to exploit virtual resources depending on the emergency or priority of the incoming tasks. As VMM we have used Xen [12], which provides useful techniques to control the VMs behavior, such as CPU limitation. The experiment presented in this work consists on evaluating how many executions of urgent incoming jobs (DDDGA) can be carried out in the same time of another task already in execution, without the need neither to suspend it nor to refuse the incoming requests, depending on the CPU limitation balance between the virtual working nodes instances. Results are shown on table 2. The time incurred by Condor to transfer the VM image is not included. table 2 only shows the actual time incurred in the execution of each job once the corresponding VM was started. It is worth noting that, according to our architecture design, each instance of a job does not require to run in a different VM. In general, only a small set of VM can be started and then multiple instances of the job will run there until the whole set of jobs is completed, thus limiting the overall overhead incurred in VM transfer and initialization. In this experiment, DDDGA and BLAST run simultaneously on the same machine but a different amount of CPU was given to each VM. The CPU amount ranged from a minimum of 10% to a maximum of 90% (for instance, first row in table 2 shows the results of DDDGA running in a VM that was limited to 10%, while BLAST was running in a VM that was using 90% of the physical CPU). The same experiment was conducted on different machines of our testbed and table 2 shows average results.

Table 2. CPU limitation experimental results (execution times in minutes)

CPU limit distribution (DDDGA VM-BLAST VM)	#DDDGA executions	#BLAST executions	Whole DDDGA set execution time	Whole BLAST set execution time	DDDGA mean execution time	BLAST mean execution time
10%-90%	1	2	128	132	128	66
25%-75%	2	1	82	82.5	41	82.5
50%-50%	5	1	126	129	25.2	129
75%-25%	18	1	277	287	15.39	287
90%-10%	49	1	591	602	12.06	602

As seen in table 2, it is possible to get an important benefit by taking into account applications behavior in each case of CPU limitation. A hypothetic situation could be the fact of having a set of non-urgent BLAST jobs queued and/or in execution, and an urgent request for executing as many instances of DDDGA as possible. The results obtained indicate that if the system balanced properly the use of CPU on the virtual resources many urgent tasks could be completed while non-urgent ones will be kept in execution, without having to suspend them. For instance, when DDDGA was running with 90% of CPU, the overhead on the execution time of each job was nearly negligible, compared to the case where DDDGA was running alone (Table 1). A total of 49 instances of DDDGA were completed at the time taken by BLAST to complete one instance. Obviously, performance degradation of BLAST was significantly higher, but this is an acceptable situation in a scenario in which an urgent application arrives into the system and needs to harness as many computational resources as possible.

Further exploitation of this multiprogramming mechanism requires the study of new scheduling policies that make an efficient use of it in order to get the maximum performance for each node in the grid site. Open issues that can be explored in the future include:

a) To establish useful scheduling policies for virtual resources exploitation.
b) To satisfy, by means of these techniques, multiple time constraints (i.e. jobs deadlines) thanks to intelligent strategies based on real-time CPU and memory limitation for jobs in execution in order to provide a suitable environment for incoming tasks, respecting both the *old jobs* and the *new jobs* deadlines.

Furthermore, the addition of some sampling mechanism will enable our system to give feedback to the users about execution time estimations, taking into account how many resources could be devoted to their jobs in a given period of time, and how would be possible to resize them in terms of CPU limitation.

5 Conclusions and Future Work

This work addresses some of the problems related to execution of urgent jobs in a distributed computing environment. We have studied and proposed an architecture for a job management service, based on virtual resources, that local sites should present in order to attend successfully and efficiently incoming computing-intensive scientific applications. By using virtualization techniques, our system has several advantages because urgent codes will be able to run without worrying about applications' libraries and architecture dependencies. The system includes separated services that are responsible, on the one hand, for managing virtual machine instantiation and, on the other hand, for job execution on the corresponding virtual machine instances. One of our goals was to keep interoperability with already existing Local Resource Management Systems (LRMS) and cluster middleware. No changes are required to existing LRMS, and the only additional software required on each cluster are the necessary components needed by the underlying virtualization engine (Xen, VMware...).

A first prototype of our service has been built and tested on a cluster managed by Condor. Several experiments were conducted to evaluate the potential benefits that can be obtained when a multiprogramming mechanism was used to share the execution of multiple virtual machines on a single host. Our experiments demonstrate that having control over virtual resources and balancing the amount of CPU devoted to each virtual machine enables the development of new scheduling strategies which could be used to manage the execution of urgent jobs by adapting the system in a way that can satisfy the incoming requests and demands.

Future research includes the exploitation of previous knowledge about applications behavior, that may allow the system to perform efficient scheduling policies [13]. Furthermore, we are working on defining and implementing additional feasible features of the architectural design proposed, which deal with issues such as automatic and dynamic VM activation according to the workload, the ability of virtual resources *booking*, or multi-core architectures exploitation in order to satisfy scientific computing intensive parallel applications.

References

1. Homepage: Spruce (November 2008), http://spruce.teragrid.org/
2. Homepage: Grid appliance user interface (November 2008), http://www.grid-appliance.org/
3. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu, X.: From virtualized resources to virtual computing grids: the in-vigo system. Future Gener. Comput. Syst. 21, 896–909 (2005)
4. Kiyanclar, N., Koenig, G.A., Yurcik, W.: Maestro-vc: A paravirtualized execution environment for secure on-demand cluster computing. In: CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, 28 (2006)
5. Webpage: Egee - glite (November 2008), http://glite.web.cern.ch/glite/
6. Pacini, F.: Jdl attributes specification (November 2008), https://edms.cern.ch/document/590869

7. Keahey, K., Foster, I.T., Freeman, T., Zhang, X.: Virtual workspaces: Achieving quality of service and quality of life in the grid. Scientific Programming 13, 265–275 (2005)
8. Keahey, K., Foster, I.T., Freeman, T., Zhang, X., Galron, D.: Virtual workspaces in the grid. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 421–431. Springer, Heidelberg (2005)
9. Homepage: Condor project (November 2008), http://www.cs.wisc.edu/condor/
10. Denham, M., Cortés, A., Margalef, T., Luque, E.: Applying a dynamic data driven genetic algorithm to improve forest fire spread prediction. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 36–45. Springer, Heidelberg (2008)
11. Homepage: Blast: Basic local alignment search tool (November 2008), http://blast.ncbi.nlm.nih.gov
12. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003), Bolton Landing, NY, USA. ACM, New York (2003)
13. Sherwood, T., Perelman, E., Hamerly, G., Calder, B.: Automatically characterizing large scale program behavior. In: Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002), San Jose, California, pp. 45–57 (October 2002)

Dynamic Software Updates for Accelerating Scientific Discovery

Dong Kwan Kim, Myoungkyu Song, Eli Tilevich, Calvin J. Ribbens, and Shawn A. Bohner

Center for High-End Computing Systems (CHECS)
Dept. of Computer Science, Virginia Tech
Blacksburg, VA 24061, USA
{ikek70,mksong,tilevich,ribbens,sbohner}@cs.vt.edu

Abstract. Distributed parallel applications often run for hours or even days before arriving to a result. In the case of such long-running programs, the initial requirements could change after the program has started executing. To shorten the time it takes to arrive to a result when running a distributed computationally-intensive application, this paper proposes leveraging the power and flexibility of dynamic software updates. In particular, to enable flexible dynamic software updates, we introduce a novel binary rewriting approach that is more efficient than the existing techniques. While ensuring greater flexibility in enhancing a running program for new requirements, our binary rewriting technique incurs only negligible performance overhead. We validate our approach via a case study of dynamically changing a parallel scientific simulation.

Keywords: Dynamic Software Updates, Time-to-Discovery, Computationally-Intensive Applications, JVM HotSwap, Bytecode Enhancement.

1 Introduction

Scientific computing is an interdisciplinary research area that uses computer technologies to analyze mathematical models for computationally demanding problems, including forecasting the weather, predicting earthquakes, and simulating molecular dynamics. Despite the ever increasing computing power, scientific computing applications are often long-running, taking hours or even days to arrive to a result, due to the tremendous amounts of involved computations. An effective approach to reducing the computing time in scientific programs is parallel processing, particularly using compute clusters and computational grids.

In a long-running application, the initial scientific requirements could change while the execution is in progress. To realize the changed requirements, a standard approach requires stopping the running application, changing the code, and restarting the application. However, this maintenance approach does not utilize the computing resources most effectively, as it leads to repeating some of the computation.

This work is concerned with perfective maintenance required to address changes in requirements rather than corrective maintenance required to address defects. We assume that the running program is correct, but needs to change to meet some newly-discovered requirements. We are not considering the problem of detecting and correcting program defects, which is addressed elsewhere in the research literature [1].

This work targets distributed computationally-intensive applications that use the Java^{TM}technology as a means to operate in heterogeneous environments. Successful applications of the Java technology to the domain of distributed parallel computation include heterogeneous, Java-based, computational grids [2]. The Java Virtual Machine (JVM) provides an advanced virtual execution environment on multiple platforms. The adaptive optimization capabilities of Just-In-Time (JIT) compilers make the JVM suitable for executing programs written in scientific computing languages, including X10 [3], and possibly other emerging languages such as Fortress [4].

Although the JVM features the HotSwap API [5], which replaces loaded classes in a running application, the signature of a replaced class must remain the same, allowing only method body changes. This, in turn, constrains the programmer modifying the swapped classes. This paper shows how these HotSwap constraints can be overcome to allow the programmer to update classes without restrictions. To that end, this paper presents a novel bytecode rewriting and code generation approach, enabling the use of the standard HotSwap to replace the changed code in a running JVM. The approach leverages *Binary Refactoring*, a technique we introduced [6] that applies semantics-preserving transformations to the binary representation of a program. The flexible and efficient dynamic updates enable the programmer to perfect a running application at will. The resulting incremental perfective maintenance model can reduce time-to-discovery when fine-tuning a distributed computationally-intensive application.

This paper presents a solution to the problem of updating computationally intensive applications dynamically and contributes a novel dynamic update method that can perfect long-running, distributed, JVM-based applications for new requirements, thereby shortening their time-to-discovery; and a new binary rewriting technique that enables the enhancement of performance-sensitive applications, with minuscule performance overhead.

The rest of this paper is structured as follows. Section 2 details our approach to updating computationally-intensive software dynamically. Section 3 evaluates the flexibility and efficiency of our approach. Section 4 compares our approach with the existing state of the art. Section 5 discusses future work directions and presents concluding remarks.

2 Updating Computationally Intensive Applications Dynamically

Next we describe our flexible and efficient dynamic software updating system for computationally intensive applications.

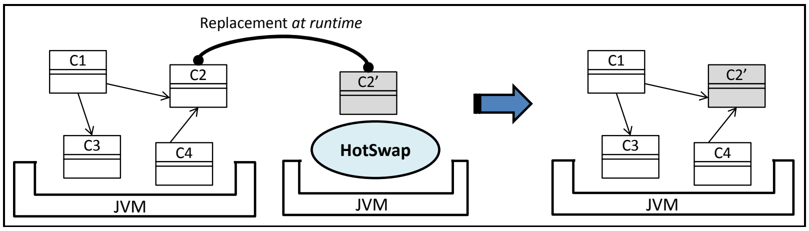


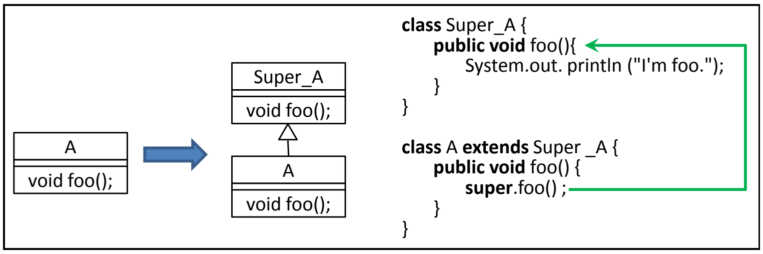
Fig. 1. JVM HotSwap facility

2.1 Enhancing JVM HotSwap Using Bytecode Rewriting

Fig. 1 shows how the JVM HotSwap reloads class *C2'* on the fly. The replaced application with four classes and the HotSwap program with a newer version *C2'* execute on two different JVMs; the JVM running the target application needs to start with the appropriate debugging options and the HotSwap module connects the JVM with its hostname and port number. The rightmost part of Fig. 1 shows the application has the new version *C2'*.

Although the HotSwap API can replace loaded classes in a running application, the signature of a replaced class must remain the same, and only method bodies could change. Thus, adding new methods, fields, or constructors, or even changing the signatures of existing methods or fields will render a class invalid for HotSwap, thus hindering the programmer from updating programs at runtime. To remove these constraints, our dynamic updating approach leverages the ability of the JVM to load classes at runtime and uses bytecode rewriting and code generation.

Fig. 2 shows our binary rewriting which introduces an indirection to a target class using the Proxy Pattern. This rewrite leverages advanced optimization capabilities of modern JVMs to inline the indirected functionality, making the rewrite applicable for performance-sensitive applications [7]. The original class *A* is translated into the proxy *A* and its superclass *Super_A*. While the class name and the method signatures of the original and proxy classes remain the same, the method bodies are different; the overloading methods of the proxy class invoke the overloaded methods of the superclass. The code snippet in the



```
class Super_A {
    public void foo(){
        System.out. println ("I'm foo.");
    }
}

class A extends Super _A {
    public void foo() {
        super.foo() ;
    }
}
```

Fig. 2. Our binary rewriting to introduce indirections

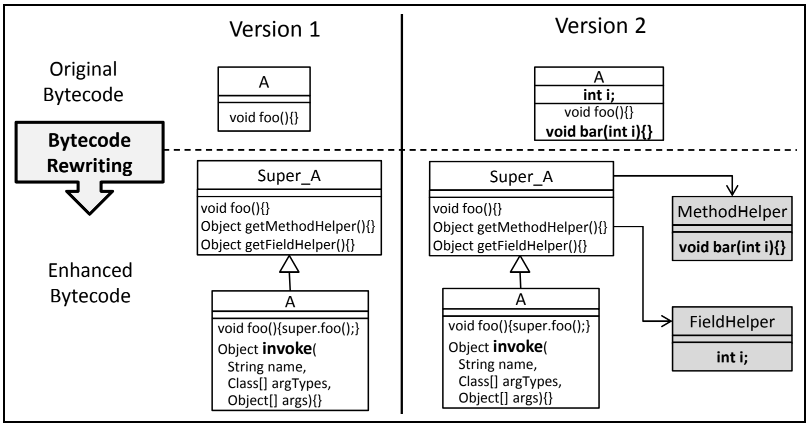


Fig. 3. Adding new members to the class *A* using the special helper classes

right shows that the proxy class *A* inherits methods from the superclass *Super_A* and the call to the method *foo()* is delegated to the superclass.

Our approach first rewrites an original program for updatability and then changes it for new requirements dynamically. In order to make a program updatable, our bytecode enhancer transforms the bytecode using the techniques described above. This approach supports a wide-range of changes to the reloaded classes, without violating the constraints imposed by the JVM HotSwap API.[1] Fig. 3 describes an example of our approach. Suppose that class *A* needs to be updated with another version. Since the second version of *A* is structurally different from the first version, the current JVM HotSwap implementation cannot reload the second version of *A*. Our approach makes new classes for added fields and methods to provide the flexibility. We called them helper classes. There are two helper classes in Fig. 3; *MethodHelper* is for the new method *bar(int i)* and *FieldHelper* is for the new field *int i*. To make intended changes to the running application, we can reload the proxy *A* and its superclass *Super_A* using HotSwap. It is obvious that JVM loads two helper classes when it reloads *Super_A*.

2.2 Updating Scientific Applications on the Fly

Once a program is made updatable at the bytecode level, the JVM HotSwap can replace at runtime the program's classes with their newer versions containing structural differences. Furher, the HotSwap facilities are used in exactly the same way as shown in Fig. 1. Fig. 4 illustrates the modules and control flow of our dynamic updating system. This system consists of the class differencing and bytecode rewriting modules.

[1] Our approach does not support the dynamic updates that change the inheritance hierarchy–these changes are too substantial to be supported by rewriting bytecode.

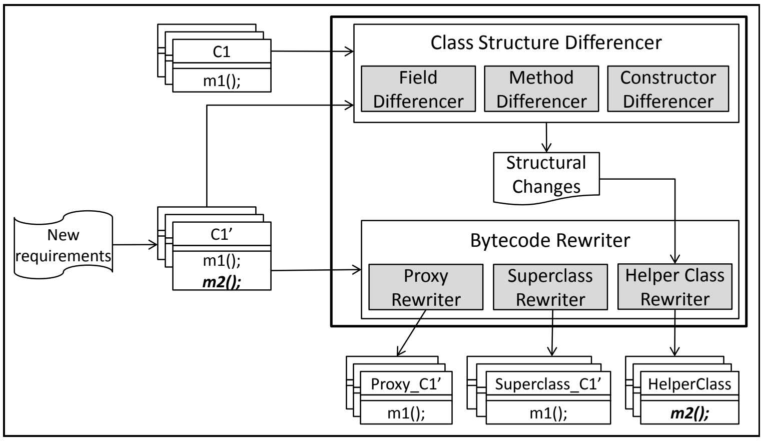


Fig. 4. Our dynamic software update system-subsystems and control flow

To generate the helper classes, our approach identifies the structural changes between the two versions of a class. The class differencing algorithm shown in Fig. 5 takes two versions of the same program as input and returns a collection of differences in fields, constructors, and methods. This algorithm simply compares the fields, methods, and constructors of the classes by using the Java Reflection API. To find out field differences, the differencing algorithm compares the modifier, type, and name of a field. The method differences are identified by examining the modifier, return type, name, and parameter types of a method and constructors are distinguished by their modifier and parameter types.

Next we provide a more formal treatment of our binary rewriting techniques using superclasses and proxies. The double vertical bar (||) specifies pre- and post-conditions. In $\frac{X}{Y}$, X denotes the class hierarchy of the original class before the enhancement, while Y denotes its new hierarchy after the enhancement has been performed. In Fig. 6, c is an original class, transformed into a proxy and with the added superclass. The new superclass c_{vs} is inserted between the proxy c_{proxy} and the initial superclass s of the original class c. Fig. 7 depicts how the indirection works for methods and constructors. Fig. 8 details how our approach introduces an indirection when accessing non-private fields. $private <_v V$ denotes the visibility V which is stronger than *private* visibility.

3 Case Study: Updating a Molecular Dynamics Simulation System Dynamically

To demonstrate the efficiency of our approach, we compared the total execution time of a Successive Over-Relaxation (SOR) [8] program with that of its rewritten version. The measurements were conducted on a compute cluster, with each node running a dual processor AMD Opteron 240 (1.4Ghz), 1GB RAM, CentOS version 4.2, JDK version 1.5.0, connected by Myrinet (4Gbit). Fig. 9 shows the total overhead of the rewritten version never exceeds 2%.

```
INPUT: A set $C = \{(c_{v1}^1, c_{v2}^1), (c_{v1}^2, c_{v2}^2), \ldots, (c_{v1}^p, c_{v2}^p)\}$ of class pairs to be compared.
OUTPUT: A collection of differences of fields, constructors, and methods
while (a set $C$ is not empty) do
  //Compare fields of classes
  $fieldsOfOldClass \longleftarrow c_{v1}^i$.allFields(), $fieldsOfNewClass \longleftarrow c_{v2}^i$.allFields()
  for ($fieldsOfNewClass$ is not empty) do
    $eachFieldOfNewClass \longleftarrow fieldsOfNewClass$.nextItem()
    for ($fieldsOfOldClass$ is not empty) do
      $eachFieldOfOldClass \longleftarrow fieldsOfOldClass$.nextItem()
      if (sig. of $eachFieldOfOldClass$ == sig. of $eachFieldOfNewClass$) then
        isSameField = true; break;
      end if
    end for
    if (NOT isSameField) then
        $differentMembers$.addElement($eachFieldOfNewClass$)
    end if
  end for
  //Compare constructors and methods of classes
  $methodsOfOldClass \longleftarrow c_{v1}^i$.allMethods(), $methodsOfNewClass \longleftarrow c_{v2}^i$.allMethods()
  for ($methodsOfNewClass$ is not empty) do
    $eachMethodOfNewClass \longleftarrow methodsOfNewClass$.nextItem()
    for ($methodsOfOldClass$ is not empty) do
      $eachMethodOfOldClass \longleftarrow methodsOfOldClass$.nextItem()
      if (sig. of $eachMethodOfOldClass$ == sig. of $eachMethodOfNewClass$) then
        isSameMethod = true; break;
      end if
    end for
    if (NOT isSameMethod) then
        $differentMembers$.addElement($eachMethodOfNewClass$)
    end if
  end for
end while
```

Fig. 5. The *ClassDifferencing* algorithm

A set of interfaces, $I = \{i_1, i_2, \ldots, i_i\}$
$VSuper(c \| c_{proxy}, c_{vs})$: Class c is transformed into c_{proxy} and c_{vs}.
c: refactored class, c_{proxy}: proxy class of c, c_{vs}: new superclass of c

$$VSuper(c \| c_{proxy}, c_{vs}) = \frac{c\ extends\ s\ implements\ I}{c_{proxy}\ extends\ c_{vs},\ c_{vs}\ extends\ s\ implements\ I}$$

Fig. 6. Indirection using superclasses

To assess the applicability of our approach to more realistic programs, we used a parallel Molecular Dynamics Simulation (MDS) program [9,10] which was deployed on Ibis [2], a Java-based grid programming environment. Among other services, Ibis provides a Java API for MPI-like message passing among cluster nodes.

```
$P_{vs}$ ||...|| denotes the rewriting by our approach.
//The transformation of constructors
$P_{vs}$ || public $k(T_1, \ldots, T_n)$ throws $C_1, \ldots, C_i$|| =
    public $k(T_1, \ldots, T_n)$ throws $C_1, \ldots, C_i$ { super($T_1, \ldots, T_n$); }
//The transformation of methods
$P_{vs}$ || public T $m(T_1, \ldots, T_n)$ throws $C_1, \ldots, C_i$|| =
    public T m($T_1, \ldots, T_n$) throws $C_1, \ldots, C_i${
        if ( the return type of $m$ is void ) super.$m(T_1, \ldots, T_n)$;
        else return super.$m(T_1, \ldots, T_n)$;
    }
```

Fig. 7. Indirecting constructors and methods

```
//Access the superclass's non-private fields
$G_{vs}$ ||...|| represents the generation of getters and setters for fields.
$G_{vs}$ || private $<_v$ V T x|| =
        private $<_v$ V T get$X$() { return $x$; }
        private $<_v$ V void set$X$(T $x$) { this.$x$ = $x$; }
//Access non-private fields via a proxy
$P_{vs}$ || private $<_v$ V T x|| =
        private $<_v$ V T get$X$() { return super.get$X$(); }
        private $<_v$ V void set$X$(T $x$) { super.set$X$($x$); }
```

Fig. 8. Indirecting the superclass's non-private fields

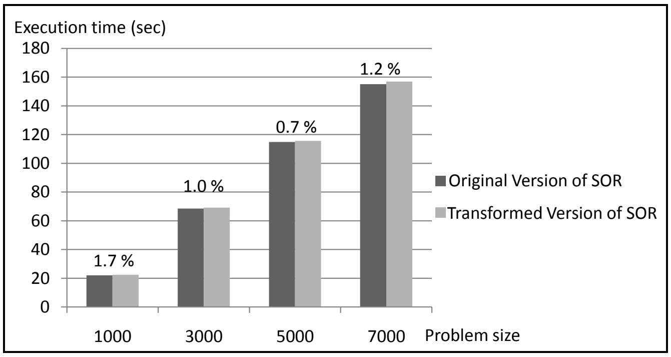


Fig. 9. Refactoring overhead on Successive Over-Relaxation. – x-axis: the problem size; y-axis: the total execution time of both the original and the enhanced versions.

We updated the MDS program dynamically twice, updating the thermostat algorithm and the number of molecules. The thermostat algorithm maintains or rescales the temperature constant of a molecular system by increasing or decreasing the velocity of the molecules. Therefore, the selection of an appropriate thermostat method depends on the molecular system in use. Also the initial

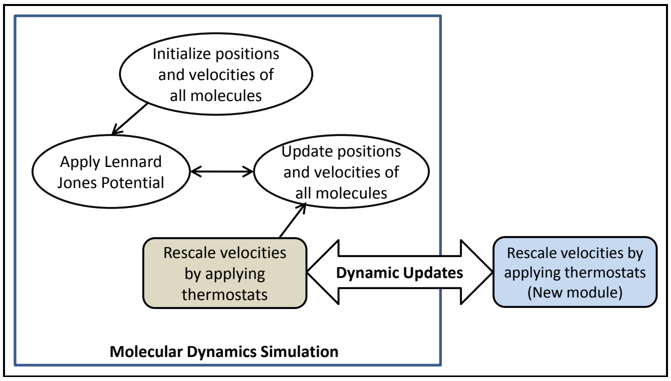

Initialize positions and velocities of all molecules
Apply Lennard Jones Potential
Update positions and velocities of all molecules
Rescale velocities by applying thermostats
Dynamic Updates
Rescale velocities by applying thermostats (New module)
Molecular Dynamics Simulation

Fig. 10. Updating the rescaling module of a molecular dynamics simulation

Table 1. Changes to the Molecular Dynamics Simulation

Updates	Requirements	Implementations
Thermostat algorithm	Rescale velocities of molecules by replacing the thermostat algorithm	Adding a new method `rescale(mol[] m, int size)` and fields
The number of molecules	Increase/decrease the number of molecules to be simulated	Adding a new method `updateNumOfMols(int size)`

number of the molecules may need to change during the simulation. Fig. 10 depicts the main modules of an MDS program and the thermostat module that are updated dynamically. Table 1 summarizes the aforementioned scenarios, which motivate dynamic changes.

While the changes above may seem simple, without dynamic updating facilities, they would require stopping the parallel execution and losing valuable computing resources. Furthermore, these updates could not be accomplished by using HotSwap alone. In fact, trying to use HotSwap for these updates would throw an exception terminating the program's execution. Finally, these changes are a natural consequence of delivering solutions under tight deadlines. It is not always possible to put enough care into designing a distributed parallel application, so that it always satisfies the requirements of different users.

4 Related Work

A significant amount research has been conducted in dynamic software updating via program transformation [11,12], custom virtual machines or runtime libraries [13,14], and new language constructs [15,16] such as Aspect-Oriented Programming (AOP) [17].

Table 2. Comparison to related work on dynamic updating for Java software(supported:+, unsupported:-, and partially supported:+/-)

Criteria	Orso[11]	Bialek[12]	Mal.[14]	Lee[21]	Pre.[18]	Ours
Use of Standard						
-Standard virtual machine	+	+	-	+	-	+
-HotSwap	-	-	-	-	-	+
-No coding constraints	+	+	+	-	+	+
-No runtime library required	+	+	-	-	-	+
Flexibility						
-Adding fields/methods	-	+	+	+	+	+
-Update of fields/methods	-	+	+	+	+	+
-No source modification	+	+	+	+	+	+
Efficient code	-	-	+	+/-	-	+

Orso *et al.*'s technique [11] and Bialek *et al.*'s system [12] transform the code to enable its dynamic updates. Unlike our approach facilitates JVM HotSwap, both approaches do not use HotSwap and consider an efficient implementation of the proxy pattern for performance-sensitive applications. Furthermore, although custom virtual machines might be more powerful in dynamic software updates, they could lead to a severe interoperability issue in a heterogeneous computing environment. Like Warth *et al.*'s Expanders [15] and Bierman *et al.*'s UpgradeJ [16], dynamic updates can be provided as new language features or a service of middleware systems. While they can express explicitly changes at the code level, the programmer is required to learn new language constructs or tools. Similar to program transformation, AOP-based approaches need to insert dynamic update modules, usually aspects, into a target application before the application is executed [18,19,20]. Table 2 compares the proposed approach with closely related work. While these approaches to dynamic updates are powerful and effective, none of them is applied and tested for computationally intensive applications such as scientific and bioinformatics programs.

5 Future Work and Conclusions

The flexibility and efficiency of our approach open a slew of future work directions, including the application of our approach to large scale grid applications, self-adapting systems, and autonomic computing.

We have presented a new binary rewriting approach for supporting flexible and efficient dynamic updates of JVM-based, distributed, computationally-intensive applications. Our approach to dynamic updating works with standard JVMs and their built-in HotSwap facility to reload classes at runtime. The performance and flexibility advantages of our approach make it promising for reducing the time-to-discovery in long-running scientific applications.

References

1. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y.: Rx: treating bugs as allergies—a safe method to survive software failures. In: SOSP 2005: Proceedings of the twentieth ACM symposium on Operating Systems Principles, pp. 235–248. ACM, New York (2005)
2. van Nieuwpoort, R.V., Maassen, J., Wrzesinska, G., Hofman, R., Jacobs, C., Kielmann, T., Bal, H.E.: Ibis: a flexible and efficient Java based grid programming environment. Concurrency and Computation: Practice and Experience 17(7-8), 1079–1107 (2005)
3. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. SIGPLAN Not. 40(10), 519–538 (2005)
4. Allen, E., Chase, D., Hallett, J., Luchangco, V., Maessen, J.W., Ryu, S., Steele Jr., G.L., Tobin-Hochstadt, S.: The Fortress language specification. Sun Microsystems, lnc. (March 2007)
5. Sun Microsystems, Inc.: Java HotSwap, `http://java.sun.com/j2se/1.4.2/docs/guide/jpda/enhancements.html`
6. Tilevich, E., Smaragdakis, Y.: Binary refactoring: Improving code behind the scenes. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, pp. 264–273. Springer, Heidelberg (2006)
7. Kim, D.K., Tilevich, E.: Overcoming JVM HotSwap constraints via binary rewriting. In: First ACM Workshop on Hot Topics in Software Upgrades. ACM, New York (2008)
8. Ortega, J.M.: Introduction to Parallel Vector Solution of Linear Systems. Plenum Press, New York (1988)
9. Rapaport, D.C.: The Art of Molecular Dynamics Simulation. Cambridge University Press, New York (1996)
10. Kumar, A.: Molecular Dynamics Simulations, `http://www.personal.psu.edu/auk183/MolDynamics/Molecularimulations.html`
11. Orso, A., Rao, A., Harrold, M.J.: A technique for dynamic updating of Java software. In: Proceedings of the International Conference on Software Maintenance (ICSM 2002) (October 2002)
12. Bialek, R.P.: Dynamic updates of existing Java applications. In: Ph.D. Thesis, the University of Copenhagen, 1–216 (2006)
13. Gharaibeh, B., Dig, D., Nguyen, T.N., Chang, J.M.: dReAM: Dynamic refactoring-aware automated migration of Java online applications. Technical Report, Iowa State University (August 2007)
14. Malabarba, S., Pandey, R., Gragg, J., Barr, E., Barnes, J.F.: Runtime support for type-safe dynamic java classes. In: Bertino, E. (ed.) ECOOP 2000. LNCS, vol. 1850, pp. 337–361. Springer, Heidelberg (2000)
15. Warth, A., Stanojević, M., Millstein, T.: Statically scoped object adaptation with Expanders. In: OOPSLA 2006, pp. 37–56 (2006)
16. Bierman, G., Parkinson, M., Noble, J.: UpgradeJ: Incremental typechecking for class upgrades. In: Vitek, J. (ed.) ECOOP 2008. LNCS, vol. 5142, pp. 235–259. Springer, Heidelberg (2008)
17. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)

18. Previtali, S.C., Gross, T.R.: Dynamic updating of software systems based on aspects. In: Proceedings of the 22nd IEEE International Conference on Software Maintenance, pp. 83–92 (September 2006)
19. Gustavsson, J., Staijen, T., Assmann, U.: Runtime evolution as an aspect. In: First International Workshop on Foundations of Unanticipated Software Evolution (2004)
20. Yang, Z., Cheng, B.H.C., Stirewalt, R.E.K., Sowell, J., Sadjadi, S.M., McKinley, P.K.: An aspect-oriented approach to dynamic adaptation. In: WOSS 2002: Proceedings of the first workshop on Self-healing systems, pp. 85–92. ACM, New York (2002)
21. Lee, Y.F., Chang, R.C.: Java-based component framework for dynamic reconfiguration. IEE Proceedings - Software 152(3), 110–118 (2005)

Generating Empirically Optimized Composed Matrix Kernels from MATLAB Prototypes

Boyana Norris[1,⋆], Albert Hartono[2], Elizabeth Jessup[3,⋆⋆], and Jeremy Siek[3]

[1] Argonne National Laboratory
`norris@mcs.anl.gov`
[2] Ohio State University
`hartonoa@cse.ohio-state.edu`
[3] University of Colorado at Boulder
{`elizabeth.jessup,jeremy.siek`}`@colorado.edu`

Abstract. The development of optimized codes is time-consuming and requires extensive architecture, compiler, and language expertise, therefore, computational scientists are often forced to choose between investing considerable time in tuning code or accepting lower performance. In this paper, we describe the first steps toward a fully automated system for the optimization of the matrix algebra kernels that are a foundational part of many scientific applications. To generate highly optimized code from a high-level MATLAB prototype, we define a three-step approach. To begin, we have developed a compiler that converts a MATLAB script into simple C code. We then use the polyhedral optimization system Pluto to optimize that code for coarse-grained parallelism and locality simultaneously. Finally, we annotate the resulting code with performance-tuning directives and use the empirical performance-tuning system Orio to generate many tuned versions of the same operation using different optimization techniques, such as loop unrolling and memory alignment. Orio performs an automated empirical search to select the best among the multiple optimized code variants. We discuss performance results on two architectures.

Keywords: MATLAB, code generation, empirical performance tuning.

1 Introduction

The development of high-performance numerical codes is challenging because performance is determined by complex interactions among the algorithm, data structure, programming language, compiler, and computer architecture. Scientists seeking high performance are thus required to master advanced concepts

⋆ The work of this author was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.

⋆⋆ The work of this author was supported by the National Science Foundation under grants no. CCF-0430646 and CCF-830458.

in computer science and carry out intricate programming tasks in addition to managing the scientific content of their work. They must either invest substantial time in tuning their software or accept low performance. In either case, the productivity of the scientists degrades.

Historically, the research community has pursued two separate paths toward the goal of making software run at near-peak levels. The first path builds on research into compilers and their associated technologies. One of the main goals of compilation research is to take an arbitrary code as input and produce optimal code as output for a given language and hardware platform. The success of this approach has been limited by a number of factors: (i) optimal mappings between the computation graph and the hardware are expensive (often NP-complete) to compute; (ii) potentially useful information that could aid optimization cannot be represented in general-purpose languages such as C and Fortran; and (iii) user control of compiler optimizations is limited and varies from compiler to compiler, and (iv) apart from differences in execution time, it is difficult to evaluate the effectiveness of different compiler optimizations.

When compilers alone cannot achieve the desired performance, another path to performance optimization is to identify kernel routines that dominate the execution time of a wide variety of applications. An example is the high-performance Basic Linear Algebra Subprograms (BLAS) libraries [1] produced by a combination of hardware vendors, independent software vendors, and researchers. Developers who write their codes calling these routines can achieve high performance across all supported architectures, but are also subject to the limitations of the library (e.g., portability and the types of operations available).

This paper describes a combination of the two approaches designed to overcome some of their shortcomings. We describe our initial efforts toward the development of software infrastructure for *generating* automatically tuned libraries for matrix algebra computations. In Section 2 we briefly discuss relevant prior and current research efforts. In Section 3 we describe our MATLAB-to-C compiler and the empirical performance tuning system Orio and its use in conjunction with the Pluto tool suite to generate and empirically evaluate many tuned versions of the C code generated by the MATLAB compiler. In Section 4 we provide performance results on two architectures. In Section 5 we conclude with a brief summary.

2 Background

Existing optimizing MATLAB [2] compilers, such as the MaJIC MATLAB compiler [3], include limited local optimizations for matrix expressions but do not perform optimizations such as loop fusion across multiple operations as we do with the tools described in this paper. The telescoping languages project [4] uses techniques such as strength reduction, vectorization, and procedure specialization to optimize MATLAB scripts but does not generate reusable optimized linear algebra routines as described in this paper.

The most common approach to tuning numerical codes is for an expert to transform the source manually, unrolling loops, blocking for multiple levels of cache, and inserting prefetch instructions. The pitfalls of this approach are well understood [5]: It requires a significant amount of time and effort. Optimizing code for one particular platform may in fact make it less efficient on other platforms and often makes it complex and hard to understand or maintain. An alternative is the use of tuned libraries of key numerical algorithms, for example, BLAS [6] and LAPACK [7] for dense linear algebra.

Specialized code generators circumvent the high costs of manual code generation. They include tools for basic dense linear algebra operations (ATLAS [8], PhiPAC [9]), and sparse linear algebra (OSKI [10]) among others. While these libraries target a specific kernel, our approach aims at enabling the definition of arbitrary kernels involving dense matrix linear algebra. It is often impossible to predict precisely the performance of code on modern computer architectures. Thus, many of these specialized code generators exploit search strategies to identify the best (or nearly best) code for a particular choice of problem parameters and machine. Most existing autotuning tools are not general but focus on a specific domain or algorithm.

A number of source or binary transformation tools for general performance-improving optimizations exist. LoopTool [11], developed at Rice University, supports annotation-based loop fusion, unroll-and-jam, skewing, and tiling. A relatively new tool, POET [12], also supports a number of loop transformations. POET offers a complex template-based syntax for defining transformations in a language-independent manner (but currently only C++ is supported). Pluto [13] is a source-to-source transformation tool for optimizing sequences of nested loops. Pluto employs a polyhedral model of nested loops, where the dynamic instance (iteration) of each statement is viewed as an integer point in a well-defined space, called the statement's polyhedron. Combined with a characterization of data dependences, this representation allows the construction of mathematically correct complex loop transformations. The transformations target both improved cache locality and parallelism.

3 Optimizing Composed BLAS Operations

Codes based on matrix algebra are generally constructed as a sequence of calls to the BLAS and similar sparse matrix libraries [14]. Writing programs in this way promotes readability and maintainability but can be costly in terms of memory efficiency. Specifically, the retrieval of a large-order matrix at each routine call can profoundly affect performance even when highly tuned implementations of the BLAS (e.g., [15]) are used.

A much more efficient approach is to call a single, specialized routine that performs multiple operations, rather than to make successive calls to separate BLAS routines (e.g., see [16]). Single routines that carry out more than one linear algebra operation are known as *composed* BLAS. As an example, consider the pair of matrix-vector products $q = Ap$, $s = A^T r$, where A is a matrix and p, q, r, and s are vectors, that represent the computational bottlenecks of the biconjugate gradient method (BiCG) [17] and of the GEMVER kernel examined in Section 3.1.

These two operations can be implemented as a pair of calls to the BLAS routine GEMV, or they can be rewritten as a composed BLAS consisting of a doubly nested loop encompassing both matrix-vector products. In the former case, the matrix A is accessed twice, whereas in the latter it is accessed only once. Our preliminary results in prior research and in the work reported in this paper indicate that loop fusion leads to a routine that delivers significantly better performance than does a pair of calls to the best optimized BLAS GEMV routines for large matrix orders. Composed routines are the focus of the work presented here, but much of what we discuss generalizes to a much broader array of computations.

To generate highly optimized code from a MATLAB prototype of the composed BLAS operation, we follow a three-step approach, illustrated in Figure 1. This process is repeated when the MATLAB code changes or the code must be tuned on a new architecture. To begin, we have developed a compiler that converts a MATLAB script into simple C code [18]. After generating the C code from the high-level MATLAB prototype, we (optionally) use the source-to-source automatic parallelization tool Pluto [13] to optimize for coarse-grained parallelism and locality simultaneously. Using the results of the Pluto analysis, we insert annotations into the C code, which are then processed by our extensible annotation system Orio to generate many tuned versions of the same operation using different optimization parameters. Orio then performs an empirical search to select the best among the multiple optimized code variants.

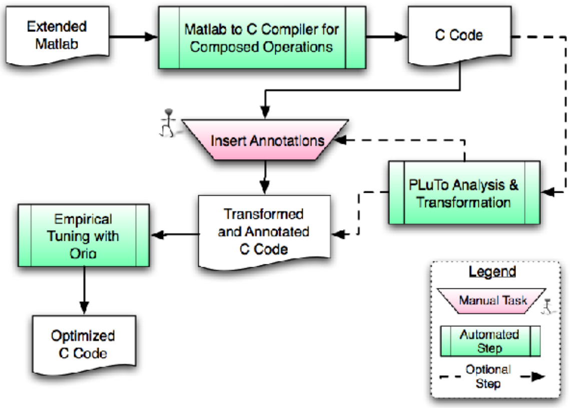

Extended Matlab
Matlab to C Compiler for Composed Operations
C Code
Insert Annotations
PLuTo Analysis & Transformation
Empirical Tuning with Orio
Transformed and Annotated C Code
Optimized C Code
Legend
Manual Task
Automated Step
Optional Step

Fig. 1. Code generation and tuning process

In the remainder of this section we describe each of the tools developed by the authors of this paper, namely, the MATLAB-to-C compiler [18] and the Orio empirical tuning tool [19,20].

3.1 A MATLAB Compiler

Figure 2 gives an overview of the MATLAB-to-C compilation process [18]. The MATLAB kernel specification is parsed into a high-level intermediate representation in the form of a dataflow graph, in which each node represents a parameter (e.g., a scalar, matrix, or vector variable) of the kernel or an operation. This dataflow graph is then iteratively processed until all of the implementation choices have been made. The compilation process consists of three phases – analysis, refinement, and optimization – that are together iterated until all of the implementation decisions have been made. The graph is then translated into C code.

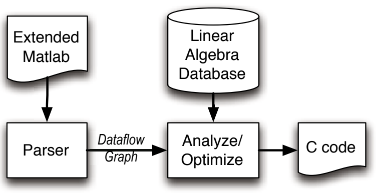


Fig. 2. MATLAB-to-C compiler

Here we briefly describe the analysis, refinement, and optimization of the dataflow graph; these are discussed in more detail in [18]. During the analysis phase, all types of intermediate nodes are computed and assigned. The algorithm choice and storage format determination are computed simultaneously. Consider, for example, the GEMVER kernel computation: $A \leftarrow A + u_1 v_1^T + u_2 v_2^T;\ x \leftarrow \beta A^T y + z;\ w \leftarrow \alpha A x$. The multiplication of u_1 and v_1^T can be implemented by iterating over rows first or over columns first, depending on how the result is used downstream in the dataflow graph. In this case, the result is added to the outer product of u_2 and v_2^T, so we still can choose either option as long as we make the same choice for both outer products.

The information on implementation possibilities for basic linear algebra operations is not hard-coded in the compiler; rather, this data is stored in a database, called the *linear algebra database*. This separation allows us to add new matrix formats, operations, and basic linear algebra algorithms without changes to the compiler algorithm.

The analysis algorithm makes implementation choices using the most-constrained-first strategy (also known as minimum remaining values) [21]. The compiler chooses the node with the fewest matching implementations (in the linear algebra database) and assigns an algorithm name to the node. If there is more than one match, the prototype compiler picks the first. This process is repeated with all remaining nodes in the graph.

The refinement phase resolves the implementation for each operation node in the graph into a subgraph defining the details of the chosen algorithm. Each subgraph is an abstract representation of the loop that implements the given operation that also contains an iteration strategy for traversing the elements of the matrix or vector. In the optimization step, we apply conditional rewrite rules to optimize the dataflow graph, for example merging two subgraphs when they share a common operand. This rule is responsible for fusing the loops of the two matrix-vector products in the GEMVER kernel. The final step performed by the MATLAB compiler when the graph cannot be refined further is the generation of C code. The generator outputs a C loop for each subgraph based on a topological sort of the graph.

3.2 Orio

Orio [19,20] is an empirical tuning tool that takes annotated C code as input, generates multiple transformed versions of the annotated code, and empirically evaluates the performance of the generated codes, ultimately choosing the best-performing version to use in production runs.

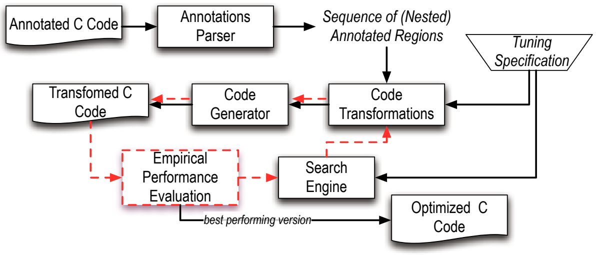


Fig. 3. Overview of the Orio empirical tuning process

Figure 3 illustrates the tuning process implemented in Orio. The input to Orio is C code containing semantic comments that describe both the computation (using a syntax that is more restricted than the original C) and various performance-tuning directives. Orio first extracts all annotation code regions by parsing the marked-up input code. Each annotated region is then passed to code transformation module and code generator for potential optimizations. Next the transformed C code with various incorporated optimizations corresponding to the specified annotations is produced. Orio generates an optimized code version for each distinct combination of performance parameter values. Each code variant is then executed and its performance measured. After iteratively testing all code variants, the best-performing code is selected as the final output of Orio. While each variant is computationally very cheap (a single code variant takes between a fraction of a second to a few seconds depending on the input sizes), the search space of all possible optimized code variants can be exponentially large. Therefore, the search engine implements a number of search heuristics (i.e., random, simplex, and simulated annealing) to effectively narrow the search for near-optimal performance and reduce the empirical search time.

Figure 4 shows an annotation example used by Orio to empirically optimize VADD operation on Blue Gene/P. The annotations contain performance hints that instruct Orio to perform memory alignment optimization, loop unrolling, and multicore parallelization (using OpenMP). In addition to these simple optimizations, Orio supports other transformations such as loop blocking, loop permutation, scalar replacement, array copy optimization, and some architecture-dependent optimizations. The right-hand side of Figure 4 shows separate tuning specifications used for building and running executable tests, including performance parameter values, execution environment details, input variable information, and the search algorithm. Orio also supports parallel search when parallel resources are available. In this example, the parallel Orio driver simultaneously executes 64 code variants in the same parallel job. At present, users must create the tuning specifications manually for each architecture. When Orio is used in conjunction with compiler tools, such as the MATLAB compiler described in this paper, it should eventually be possible to automatically generate the tuning specifications.

```
void vadd(int n, double *y, double *x1,
          double *x2, double *x3) {
/*@ begin PerfTuning(
  import spec vadd_tune_spec;
) @*/

 register int i;

/*@ begin BGP_Align(y[],x1[],
                    x2[],x3[]) @*/
/*@ begin Loop(
 transform Unroll(ufactor=UF,
                  parallelize=PAR)
 for (i=0; i<=n-1; i++)
   y[i] = x1[i] + x2[i] + x3[i];
) @*/

 for (i=0; i<=n-1; i++)
   y[i] = x1[i] + x2[i] + x3[i];

/*@ end @*/
/*@ end @*/
/*@ end @*/
}
```

```
spec vadd_tune_spec {
 def build {
  arg build_command = 'mpixlc -O3 -qstrict -lm';
  arg batch_command = 'qsub -n 64 -t 10';
  arg status_command = 'qstat';
  arg num_procs = 64;
 }
 def performance_params {
  param UF[] = range(1,32);
  param PAR[] = [True, False];
 }
 def input_params {
  param N = [10,100,1000,10**4,10**5,10**6,10**7];
 }
 def input_vars {
  decl int n = N;
  decl double y[N] = 0;
  decl double x1[N] = random;
  decl double x2[N] = random;
  decl double x3[N] = random;
 }
 def search {
  arg algorithm = 'Exhaustive';
 }
}
```

Fig. 4. Orio example: Annotated C source code (left) and tuning specification excerpt for the Blue Gene/P (right)

4 Experimental Results

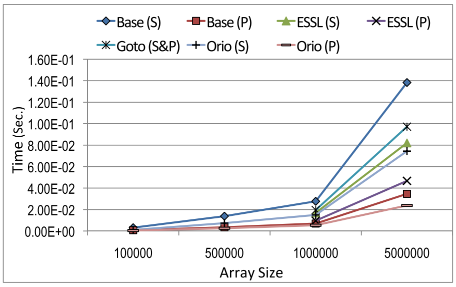
We evaluated our approach by running experiments on an Intel Xeon workstation and the Blue Gene/P at Argonne. The Intel machine has dual quad-core E5462 Xeon processors (8 cores total) running at 2.8 GHz (1600 MHz FSB) with 2 GB RAM, running Ubuntu 8.04. Intel C compiler (v10.1) was used with `-O3` option (and `-parallel`/`-openmp` for automatic/manual parallelization, respectively). Each node of the Blue Gene/P has four 850 MHz PowerPC 450 processors with a dual floating-point unit and 2 GB total memory per node, running a proprietary operating system. On the Blue Gene/P, we used IBM XLC compiler (v9.0), with `-O3 -qstrict -qarch=450d -qtune=450 -qhot` options (and `-qsmp=auto`/`-qsmp=noauto` for automatic/manual parallelization, respectively).

Table 1 lists the composed BLAS operations used in our experiments, along with their input and output variables. Vectors are typeset in lowercase with an overhead arrow. Scalars and matrices are represented as lowercase and uppercase letters, respectively. A regular uppercase denotes a row matrix, whereas a bold uppercase symbolizes a column matrix. The extended MATLAB expression that corresponds to each operation can be seen in the last column of Table 1.

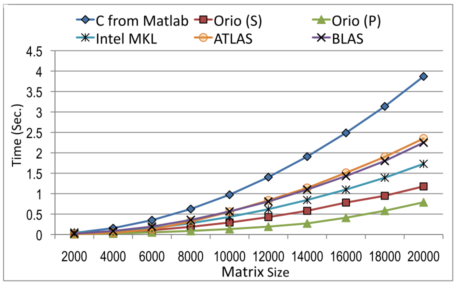
The performance results of tuning the VADD operation on the Blue Gene/P are given in Figure 5(a). The "Base" label designates the C implementation generated by the MATLAB compiler. We also tested the performance of an implementation that calls `DAXPY` twice using available BLAS libraries. Finally, we tuned the simple C loop version using Orio, with the performance annotations previously shown in Figure 4. In this experiment, we measured the performance

Table 1. Composed BLAS operations used in our experiments

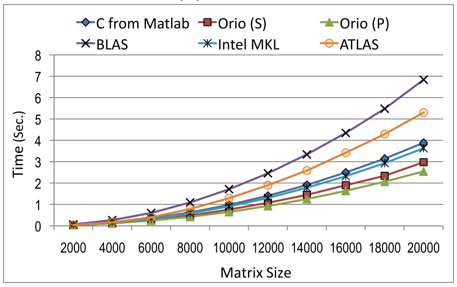
Name	Input	Output	Operation
VADD	$\overrightarrow{w}, \overrightarrow{y}, \overrightarrow{z}$	$\overrightarrow{x}$	$\overrightarrow{x} = \overrightarrow{w} + \overrightarrow{y} + \overrightarrow{z}$
ATAX	$A, \overrightarrow{x}$	$\overrightarrow{y}$	$\overrightarrow{y} = A^T * (A * \overrightarrow{x})$
GEMVER	$\mathbf{A}, a, b, \overrightarrow{u_1}, \overrightarrow{u_2}, \overrightarrow{v_1}, \overrightarrow{v_2}, \overrightarrow{y}, \overrightarrow{z}$	$\mathbf{B}, \overrightarrow{x}, \overrightarrow{w}$	$\mathbf{B} = \mathbf{A} + \overrightarrow{u_1} * \overrightarrow{v_1}^T + \overrightarrow{u_2} * \overrightarrow{v_2}^T$ $\overrightarrow{x} = b * (\mathbf{B}^T * \overrightarrow{y}) + \overrightarrow{z}$ $\overrightarrow{w} = a * (\mathbf{B} * \overrightarrow{x})$
BiCG Kernel	$\mathbf{A}, \overrightarrow{p}, \overrightarrow{r}$	$\overrightarrow{q}, \overrightarrow{s}$	$\overrightarrow{q} = \mathbf{A} * \overrightarrow{p}$ $\overrightarrow{s} = \mathbf{A}^T * \overrightarrow{r}$



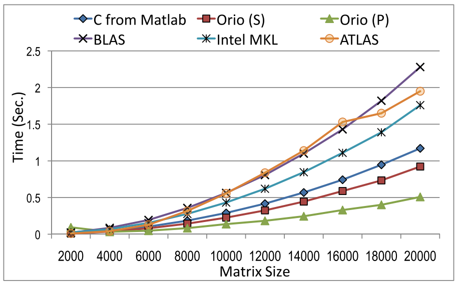
(a) VADD



(b) ATAX



(c) GEMVER



(d) BiCG Kernel

Fig. 5. Performance results for several composed BLAS operations

for both the sequential and parallel scenarios (indicated by (S) and (P) in the legend, respectively). Even for a very simple operation such as vector addition the compiler alone is unable to obtain the same level of performance as the empirically tuned versions. Furthermore, as expected, the BLAS implementation does not exploit locality and thus performed worse than the single-loop implementation.

The experiments of the remaining operations were performed on the multicore Intel Xeon. Included in these experiments are performance numbers for six code variants: the C code generated by the MATLAB compiler ("C from MATLAB"), three BLAS-based implementations that use Intel MKL, ATLAS, and the default BLAS library on Ubuntu 8.04, and the sequential and parallel code variants tuned by Orio ("Orio (S)" and "Orio (P)", respectively).

The Xeon performance results of ATAX are shown in Figure 5(b). The Orio-tuned version that incorporates Pluto-generated loop fusion optimizations and Orio parallelization directives achieves the best performance for most problem sizes, outperforming the Intel MKL version by a factor of 2 to 5.7 and the compiler-optimized C version by a factor of 4 to 7. The optimizations performed by both Orio versions include scalar replacement, vectorization, and loop unroll/jam.

Figure 5(c) shows the performance of the GEMVER operation on the Xeon workstation. Here we used the same Pluto and Orio optimizations as for the ATAX example. Similarly, the parallel Orio version achieved the best performance, although in this case the sequential Orio version performs almost the same, suggesting that the compiler was not able to parallelize the code very effectively. For this operation, substantial performance differences exist between among the different BLAS versions, with the Intel MKL version achieving performance close to that of the simple compiler code.

The performance for the BiCG kernel operation is shown in Figure 5(d). For this operation, the Pluto analysis did not result in performance improvement. Thus we are showing the results obtained only through Orio transformations, which included vectorization, scalar replacement, and loop unroll/jam. Again the best performance was achieved by the parallel Orio version, while all the BLAS versions performed worse than the compiler-optimized C loop version.

5 Conclusions

We have described an approach to generating tuned linear algebra libraries from high-level annotated MATLAB code that involves a suite of tools to (1) translate the MATLAB code to C, (2) analyze the resulting loops and identify locality and parallelism-enhancing optimizations using Pluto, and (3) annotate the resulting C code with syntactic performance directives and use Orio to generate multiple optimized versions and empirically select the one with the best performance. Preliminary results from experiments with several composed BLAS operations show that the optimized code generated by this suite of tools significantly outperforms the versions using tuned BLAS and aggressive compiler optimizations.

The positive initial results from our approach to generating tuned linear algebra routines motivate several future lines of investigation, including closer integration between the tools handling the different steps of the process and more automation at each step.

References

1. Dongarra, J.J., Croz, J.D., Duff, I.S., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Soft. 16, 1–17 (1990)
2. MathWorks: MATLAB - The Language of Technical Computing, http://www.mathworks.com/products/matlab/
3. Menon, V., Pingali, K.: High-level semantic optimization of numerical codes. In: Proceedings of the 13th International Conference on Supercomputing, pp. 434–443. ACM Press, New York (1999)
4. Kennedy, K., et al.: Telescoping languages project description (2006), http://telescoping.rice.edu
5. Goedecker, S., Hoisie, A.: Performance optimization of numerically intensive codes. Software Environments & Tools 12 (2001)
6. Dongarra, J.J., Croz, J.D., Duff, I., Hammarling, S.: A set of Level 3 Basic Linear Algebra Subprograms. ACM Trans. Math. Softw. 16(1), 1–17 (1990)
7. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK Users' Guide, 2nd edn. SIAM, Philadelphia (1995)
8. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimization of software and the ATLAS project. Parallel Computing 27(1–2), 3–35 (2001)
9. Bilmes, J., Asanovic, K., Chin, C.W., Demmel, J.: Optimizing matrix multiply using PHiPAC: A portable, high-performance, ANSI C coding methodology. In: International Conference on Supercomputing, pp. 340–347 (1997)
10. Vuduc, R., Demmel, J., Yelick, K.: OSKI: A library of automatically tuned sparse matrix kernels. In: Proceedings of SciDAC 2005. Journal of Physics: Conference Series, vol. 16, pp. 521–530. Institute of Physics Publishing (June 2005)
11. Fowler, R., Jin, G., Mellor-Crummey, J.: Increasing temporal locality with skewing and recursive blocking. In: Proceedings of SC 2001: High-Performance Computing and Networking (November 2001)
12. Yi, Q., Seymour, K., You, H., Vuduc, R., Quinlan, D.: POET: Parameterized optimizations for empirical tuning. In: Proceedings of the Parallel and Distributed Processing Symposium, 2007, pp. 1–8. IEEE, Los Alamitos (2007)
13. Bondhugula, U., Hartono, A., Ramanujam, J., Sadayappan, P.: Pluto: A practical and fully automatic polyhedral program optimization system. In: Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI 2008), Tucson, AZ (June 2008)
14. Saad, Y.: SPARSKIT: A basic tool kit for sparse matrix computations. University of Minnesota, Department of Computer Science and Engineering (1990)
15. Goto, K., van de Geijn, R.: High-performance implementation of the level-3 BLAS. Technical Report TR-2006-23, The University of Texas at Austin, Department of Computer Sciences (2006)
16. Gropp, W.D., Kaushik, D.K., Keyes, D.E., Smith, B.F.: High-performance parallel implicit CFD. Parallel Computing 27, 337–362 (2001)
17. Saad, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2003)
18. Jessup, E., Karlin, I., Siek, J.: Build to order linear algebra kernels. In: Proceedings of the IEEE International Symposium on Parallel and Distributed (IPDPS), pp. 1–8. IEEE, Los Alamitos (2008)

19. Norris, B., Hartono, A., Gropp, W.: Annotations for productivity and performance portability. In: Petascale Computing: Algorithms and Applications. Computational Science, pp. 443–462. Chapman & Hall / CRC Press, Taylor and Francis Group (2007)
20. Hartono, A., Norris, B., Sadayappan, P.: Annotation-based empirical performance tuning using Orio. In: Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium, Rome, Italy, IEEE, Los Alamitos (2009)
21. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Inc., Englewood Cliffs (2003)

Automated Provenance Collection for CCA Component Assemblies

Kostadin Damevski and Hui Chen

Virginia State University
Petersburg, VA, 23806, USA
{kdamevski,hchen}@vsu.edu

Abstract. The problem of capturing provenance for computational tasks has recently received significant attention, due to the new set of beneficial uses (for optimization, debugging, etc.) of the recorded data. We develop a provenance collection system aimed at scientific applications that are based on the Common Component Architecture (CCA) that alleviates scientists from the responsibility to manually instrument code in order to collect provenance data. Our system collects provenance data at the granularity of component instances, by automatically recording all method invocations between them, including all input and output parameters. By relying on asynchronous communication and using optimizations to handle large data arrays, the overhead of our system is low-enough to allow continuous provenance collection.

1 Introduction

Provenance is a collection of intermediate data that explains in some level of detail the transition from input data to output data in a scientific application. This type of data collection also exists in many other computing domains, under different constrains and application requirements; provenance collection is similar to logging in operating systems and to preservation of data lineage in databases. The form and granularity of data provenance depends on the type of application that is intended to consume this stored data. The collected data can be used for purposes ranging from error detection and debugging, to optimization by removing redundant computations, and even to accountability of individual parts in a multi-user system [1,2]. In the scientific computing domain, provenance also serves the role of a "paper-trail" , in concert with the source code, to document the computational methods used to a particular scientific discovery.

In recent years, component technology has been a successful methodology for large-scale commercial software development. Component technology encapsulates a set of frequently used functions into a component and makes the implementation transparent to the users. Application developers typically use a group of components, connecting them to create an executable application. Component technology is becoming increasingly popular for large-scale scientific computing in helping to tame the software complexity required in coupling multiple disciplines, multiple scales, and/or multiple physical phenomena. The Common

Component Architecture (CCA) [3] is a component model that was designed to fit the needs of the scientific computing community by imposing low overhead and supporting parallel components. CCA has already been used in several scientific domains, creating components for large simulations involving accelerator design, climate modeling, combustion, and accidental fires and explosions [4]. These types of applications usually contain a large number of connected components, each of them at the granularity of one numerical computation. As individual components are often contributed by separate teams and reused for several applications, many of them are treated as black boxes. Applications based on the CCA model can leverage data provenance to establish whether a particular component behaves properly and to localize bugs and inconsistencies in application development. In addition, we can use the collected data to prune computations that do not have side-effects when there is a match in the input data, by supplying the already recorded output data. This strategy greatly improves performance and it has been employed, outside of the component space, by computational studies, which repeatedly execute the same application to explore its the parameter space [5]. We design a system to collect provenance data in CCA applications with these uses of the data in mind.

In order to collect provenance data, one needs to instrument the application by inserting invocations to a serialization routine. This can be a time-consuming and repetitive task, and a perfect candidate for automation. Automatic instrumentation makes it easy to start collecting data for any CCA application, even if the application consists of many individual component instances. We choose to collect data at the boundary between components, capturing and recording all communication flow between a pair of components, including method invocations and associated input parameters, output parameters and return values. The component boundary is an appropriate place to collect data for the majority of CCA applications we have encountered because the collected data is usually of acceptable granularity to be used together with the component instances in order to enable provenance applications such as accountability, debugging, and the removal of redundant computation (where the computation was previously completed and its input and output data were recorded). It is also the only place where automatic instrumentation is easily attainable. In this paper, we describe our implementation of a method to automatically instrument CCA components in order to collect provenance data. Our goals in designing our system for collecting provenance are to: 1)capture all the provenance data that passes between component instances and 2)impose low overhead so that provenance will be collected continuously, even in deployment scenarios.

We organize the discussion of our provenance gathering system as follows. Section 2 contains background and discussion of the problem. In Section 3 we show a detailed view of our design and implementation of the CCA provenance collection framework, while in Section 4 we discuss some of the preliminary benchmarks we have taken of our system. Finally, we finish with conclusions and future work of the project in Section 5.

2 Background and Problem Specification

Provenance is a technique in wide use in both core computer science and computer application domains. It can be most generally explained as the preservation of metadata regarding a specific data product. Simmhan et al.'s summary of provenance in e-Science [6] establishes a taxonomy of provenance techniques based on the following criteria: 1) application of provenance, 2) subject of provenance, 3) representation of provenance, 4) provenance storage and 5) provenance dissemination. These criteria provide a vehicle for examining a provenance system, considering both provenance collection and provenance querying and use. In our work we only consider the collection of provenance data, while dissemination and use of this data is outside of the scope of this paper.

The CCA model consists of a framework and an expandable set of components. The framework is a workbench for building, connecting and running components. A component is the basic unit of an application. A CCA component consists of one or more ports, and a port is a group of method-call based interfaces. There are two types of ports: **uses** and **provides**. A provides port (or callee) implements its interfaces and waits for other ports to call them. A uses port (or caller) issues method calls that can be fulfilled by a type-compatible provides port on a different component. A CCA port is represented by an interface, which is specified through the Scientific Interface Definition Language (SIDL). A SIDL specification is compiled into glue code that is later compiled into an executable together with the user-provided implementation code. The prevalent way of compiling SIDL is by using the Babel compiler [7]. Babel has the capability of compiling SIDL to bindings for several popular programming languages (C++, Java, Python, Fortran77 and Fortran95), which allows creating applications by combining components written in any of these languages. Babel has a large and growing user community and has become a cornerstone technology of the CCA component model.

Component technology is different in its approach compared to scientific workflows and grids. Scientific workflows and grid services are intended to guide an application from the highest level combining few complex tasks, while component applications are decomposed into many finer-grain tasks. This makes the demands of a provenance system applied to CCA unique, compared to existing approaches in other scientific software domains, in terms of quantities of collected data and common usage scenarios.

Collecting provenance data imposes some overhead on the execution of an application, and if this overhead is high, it may be tempting to turn off provenance collection in order to "squeeze out" more performance from of the machine. One of the principal goals of our system is to keep the overhead low in order to enable continuous provenance collection. In order to accomplish this goal, we need to have the provenance collecting system work in the background throughout the application runtime (as a daemon), and send data to it asynchronously. In addition, scientific applications that are based on CCA often handle large pieces of data; it is not uncommon for an application to have multidimensional arrays that are hundreds of megabytes or even gigabytes in size. When designing provenance collection infrastructure, we need to be careful to avoid in-memory copies of large data, and eliminate needless performance overhead.

3 System Design

We designed and implemented a provenance collection system for CCA applications. Our design provides automatic code instrumentation with provenance calls that incur low application execution overhead. To automate the instrumentation process we need to add provenance collection code to the stub code of each CCA component, and to accomplish this we add functionality to Babel. The inserted provenance collection code will execute before and after every SIDL-defined method, gathering all the input parameters (*in* and *inout*) before the method executes and all the output parameters (*out*, *inout* and method return parameters) post execution. For instance, a SIDL definition of a CCA port containing only one method is given below. To properly collect the provenance data of the component which provides the *IntegratorPort*, we need to capture the two *in* parameters passed to *integrate* before the method executes, and the return value (of type *double*) after the execution completes.

```
interface IntegratorPort extends gov.cca.Port {
  double integrate(in double lowBound, in double upBound);
}
```

The SIDL language is relatively restrictive; it does not allow class variables and forces everything to be expressed in terms of interfaces, classes and methods. Therefore, the data types we are concerned in recording consist of: objects, SIDL-defined base types and arrays of base types. We designed our system to record each of these types, with differing levels of ease.

3.1 Automating Provenance Collection

Once properly modified, the Babel compiler creates special stubs for each method of an instrumented CCA port. These stubs copy the input and output parameters of a method, write them into a message which is later saved to disk as provenance. Since SIDL base types can only be defined as parameters of methods in the language, it is relatively straightforward to generate provenance collection code for each allowable base type (e.g. int, long, float, double, fcomplex, dcomplex, string, bool). On the other hand, this makes the serialization of objects passed as method parameters difficult to automate. Our approach is to force objects that can be serialized to implement a *Serializable* interface. Through this interface, each object must define how it can be written to disk and also reconstructed from its disk image. Our instrumented stubs detect whether an object implements *Serializable*, and will only then invoke the proper serialization methods on the object. Objects that do not know how to serialize themselves are ignored for provenance collection.

SIDL also defines an *opaque* type, typically used to represent a pointer to memory. Babel defines *opaque* as a sequence of bits that are relayed exactly between function caller and callee, which maps to a *void** in C/C++. In order for our system to capture the data to which this pointer points to, we require an additional parameter defining the length of the data. Extra parameters can be defined with SIDL and passed into the Babel compiler using the *%attrib* keyword. We use this mechanism to specify the length of the data pointed to by an *opaque* type in SIDL, and enable our provenance system to record it. For instance, using our integrator example from before, we may have:

```
interface IntegratorPort extends gov.cca.Port {
  void integrate(in opaque %attrib{length=20} data);
}
```

In the scientific domain, arrays are the predominant data type used to represent various data (e.g meshes, fields, matrices, etc.) Unlike *opaque* types, the length of an array does not need to be externally specified because it is part of each array's "struct" in Babel, and is easily obtainable through method calls available in the Babel runtime. Since arrays consist of sequences of SIDL base types, they appear straightforward to handle by our provenance system. However, some arrays in this domain can grow to millions (or more) of elements, so we need to be careful in designing our system to handle them. Below, we propose some prescriptions and optimizations in provenance collecting large CCA arrays.

In order to gather all of the provenance data in one place, as well as provide a single location for querying of such data, which can happen even as it is being written to disk, we need a provenance collection component. Such a component would provide us with a well defined interface and encapsulation for both provenance storage and query. To communicate data asynchronously to the provenance collection component we use an event mechanism. This publish/subscribe asynchronous communication mode allows the provenance component to subscribe to one or more topics in which running components can deposit provenance information. Events decouple the provenance sending from the provenance receiving, writing to disk (or database or other medium) across space and time, enabling a low overhead design. To deliver data to the provenance component, our instrumentation code serializes all the parameters going in and out of a method into a message and publishes it. The provenance component contains a daemon thread that periodically grabs all published messages and writes them to disk. Fig. 1 illustrates an example of this type of communication in our provenance gathering system.

3.2 Optimizations for Large Data

Arrays are often encountered in CCA applications, and some of them can grow to contain a considerable amount of data. CCA components are most often at the granularity of one task, and often a single application passes large arrays to several components. In a shared address space, these arrays are passed by

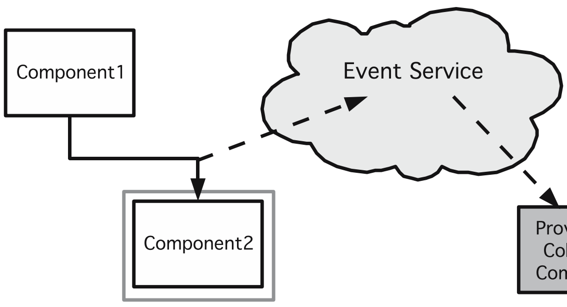


Fig. 1. Displays an example of two application components (Component1 and Component2) and our provenance collection component. When Component1 invokes a method on Component2, our system publishes the method's input parameters using an event message, which is later collected by the provenance collection component.

reference at very little performance cost, so a single method may be invoked many times with a large array as a parameter. Our system is designed to capture all data going in and out of a method invocation, serializing and pushing all this data to disk. In the case of these large arrays, this may be a task that can badly influence the performance and scalability of provenance collection.

One operation we need to avoid is in-memory copies of large arrays. If an array is sufficiently large, an in-memory copy may exceed physical memory size and may lead to OS thrashing. In order to record large arrays properly, we have to circumvent our system's default operation: copying of memory to construct an event message. We add a synchronous way to directly communicate to the provenance component in order to serialize large arrays directly to disk. We need a synchronous call for this operation as we need to ensure that the array is not modified by a component before our system finishes writing it for provenance. We only rely on this mechanism if we detect that an array is too large to copy, leaving unchanged the base system of asynchronously sending data via publish/subscribe for smaller arrays.

Another scenario we want to avoid in the context of large arrays is needlessly serializing the same array that may appear in multiple method invocations (to the same method or a different one). For instance, a visualization component may be invoked every n seconds passing to it the latest data, which may often be unchanged, or a component may simply forward a received array to another component which will do all the work. These are patterns in which our provenance collection system would make several copies of a large data array, costing us application performance and disk space. In order to avoid this scenario we use a very fast hash function to compute a checksum that will enable us to quickly and accurately compare two arrays and determine if they are the same. If two SIDL arrays are in-fact the same array, then their resulting checksums will match and we can avoid the space and time cost of storing one of them. Storing a checksum requires a very small amount of space (usually between 8 and 128 bits), however computing it requires that we perform an operation across the length of the array. We dismiss the option of computing a checksum using a portion of the array, as it runs a risk of checksum collision. Obtaining the same checksum for different arrays would cause our system not to store an array, which would result in incorrect provenance data and is something we must avoid. Although the cost of computing a checksum may take on the order of seconds for a large array, it is still several fold less than the time needed to store such an array to disk, while also not considering the aforementioned storage space savings.

4 Results

To show the feasibility of our approach, and not as an end in themselves, we performed some experiments. These proof of concept scenarios explored the choices of hash functions for the redundant copy avoidance optimization in large arrays and explored the overhead of our provenance system for a simple application: solving a boundary condition problem. All experiments were performed on a single machine with an 2 GHz Intel Core 2 Duo processor and 1GB of RAM. We made modifications to the Babel compiler provided for automatic insertion of the instrumentation code, and used SCIJump [8], a research CCA component framework that provides low overhead and support for parallel and distributed computing, to conduct our experiment.

4.1 Hash Function Selection

In choosing a hash function to optimize the recording of large arrays we should consider the tradeoff between its performance and the likelihood of collisions in the resulting checksums. Since one of the purposes of this optimization is to reduce overall application overhead, we have to consider the performance cost of hash function computation for a large data array. We cannot afford commonly used hash functions (such as MD5 or SHA) because of the enormous computational load in computing a checksum based on the whole array. While performance is very important, we cannot risk raising the probability of hash function collisions. A strategy we adopt in order to reduce the likelihood of collisions is to make the computed checksum itself larger; a larger checksum reduces the likelihood of collisions by several-fold.

To locate a reasonable and fast hash function, we considered some of the performance benchmarks of hash functions in the Crypto library [9], but concluded that all of the functions in this commonly used library are too expensive. The work of Maxino [10] evaluates and compares the speed and collision potential of very fast hash function for embedded system design, and recommends one's complement addition (also know as the Internet Checksum) as the most reasonable choice in fast hash functions, compared to commonly used XOR, and two's complement addition. To further reduce the likelihood of collision in the checksum, while not greatly increasing computational time, we extended the one's complement addition algorithm's checksum size to 64-bits for our provenance system.

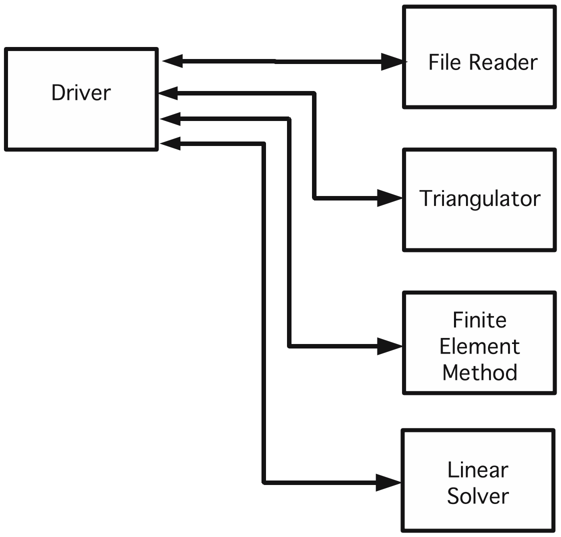


Fig. 2. The heat distribution component application for which we collected data provenance using our system. The Driver component communicates to and from each of the component instances implementing computational tasks.

In this way, the checksum computation produces a small additional time overhead in our system, while providing us with reasonable confidence that checksum collisions will not occur.

4.2 Provenance Overhead in Applications

To validate our design and determine the overall overhead, we implemented our provenance system and used it to collect the provenance of an application. The application we chose solves the heat distribution problem over a small L-shaped 2D domain. It consists of several components, each of which computes a part of the solution (see Fig. 2).

We measured that the provenance collection added less than 1% overhead to the execution time of our application, which is in line with the goals we set forth. We note that this application did not need any of the large data optimizations which we designed, due to the small size of its computational problem. The application did, however, record data containing most SIDL-defined types, resulting in approximately 1KB/method invocation of data logged.

It is possible to design applications where our kind of provenance collection would incur a larger overhead; if the problem is decomposed into a very fine grain resulting in frequent inter-component communication (method calls) that do not spend much time computing. Each method call and its parameters would be recorded by our system, raising the overhead percentage to a higher level than we have encountered here. Although this is possible, it is not the usual way that CCA (or components in general) is applied to a scientific problem.

5 Related Work

The need for data provenance has been widely acknowledged and is evident is many computer domains. Here we intend to only review systems similar to our provenance collection scheme for software components. The provenance survey by Simmhan et al. [6], provides a good overview of provenance applications across different domains. We direct the reader's attention again to this survey for an overview of provenance activities unrelated to scientific software architectures (such as components, workflows, and web services).

The Karma framework [11] collects provenance in the context of web service scientific applications. This provenance service is general and supports different web service systems by using event (publish-subscribe) standards for web services (WS-Eventing). This is the same communication mechanism, in the software component rather than web service domain, that we chose to use in our system. Our work extends Karma in providing an automated approach for provenance collection. Karma further addresses provenance query and visualization, which we do not attempt in our work so far.

Scientific workflows are a software architecture similar to components; workflows usually encompass a wider variety of tasks (such as database data collection, batch job system control etc.) and decompose a problem in a coarser grain than CCA. Altintas et al. present a provenance collection system [12] that requires minimal user effort and a system for "smart" re-runs which mine the provenance information for repetitive data. The provenance data in their system is collected by a special provenance recorder which listens and collects events that are produced by the workflow by default. However, this mechanism does not handle external data automatically, requiring special API calls. External data is probably the majority of the interesting data in workflow applications. Therefore, the provenance collection part of their work would likely require a fair amount of non automatic instrumentation of applications.

6 Conclusions and Future Work

This paper presents a design for automated provenance collection in CCA component applications that requires no user intervention and provides low overhead ($\tilde{1}$% in the application we tested) in order to be useful in deployment scenarios. We designed our provenance collecting system to record data being communicated through each component instance's ports and interfaces. Optimizations were necessary in order to handle large data arrays and the way they are often communicated between component instances. We posit that provenance data collected in this way is usable for debugging, accountability of untrusted component instances, as well as optimizations by removing computations for which we have previously gathered the output.

The future work of this project is to explore techniques of even further reducing the overhead that our system imposes on CCA applications in two ways: 1)by considering novel ways to overlap computation and collection of provenance data, and 2)by extending our approach to encompass new applications that may have a different computation to communication ratios. Each of these advances in our system is important in achieving broad applicability and acceptance in the CCA and applications communities.

References

1. Guo, Z., Wang, X., Tang, J., Liu, X., Xu, Z., Wu, M., Kaashoek, M.F., Zhang, Z.: R2: An application-level kernel for record and replay. In: Proceedings of the 8th Symposium on Operating Systems Design and Implementation (2008)
2. Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J., Silva, C.T.: Querying and creating visualizations by analogy. IEEE Transactions on Visualization and Computer Graphics (2007)
3. Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B.: Toward a common component architecture for high-performance scientific computing. In: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computation (HPDC) (1999)
4. McInnes, L.C., Allan, B.A., Armstrong, R., Benson, S.J., Bernholdt, D.E., Dahlgren, T.L., Diachin, L.F., Krishnan, M., Kohl, J.A., Larson, J.W., Lefantzi, S., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., Zhou, S.: Parallel PDE-based simulations using the Common Component Architecture. In: Bruaset, A.M., Tveito, A. (eds.) Numerical Solution of PDEs on Parallel Computers. Lecture Notes in Computational Science and Engineering (LNCSE), vol. 51. Springer, Heidelberg (2006)
5. Yau, S.M., Damevski, K., Karamcheti, V., Parker, S.G., Zorin, D.: Result reuse in design space exploration: A study in system support for interactive parallel computing. In: Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS) (2008)
6. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Record 34(3) (2005)
7. Kohn, S., Kumfert, G., Painter, J., Ribbens, C.: Divorcing language dependencies from a scientific software library. In: Proceedings of the 10th SIAM Conference on Parallel Processing, Portsmouth, VA (2001)
8. Zhang, K., Damevski, K., Venkatachalapathy, V., Parker, S.: SCIRun2: A CCA framework for high performance computing. In: Proceedings of The 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (2004)
9. Dai, W.: Crypto++ 5.5 Benchmarks (2008), http://www.cryptopp.com/benchmarks.html
10. Maxino, T.C.: The effectiveness of checksums for embedded networks. Master's thesis, Carnegie Mellon University (2006)
11. Simmhan, Y.L., Plale, B., Gannon, D.: A framework for collecting provenance in data-centric scientific workflows. In: International Conference on Web Services (2006)
12. Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance collection support in the kepler scientific workflow system. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 118–132. Springer, Heidelberg (2006)

Modular, Fine-Grained Adaptation of Parallel Programs

Pilsung Kang[1], Naresh K. C. Selvarasu[2], Naren Ramakrishnan[1], Calvin J. Ribbens[1], Danesh K. Tafti[2], and Srinidhi Varadarajan[1]

[1] Department of Computer Science, Virginia Tech, Blacksburg, VA 24061
[2] Department of Mechanical Engineering, Virginia Tech, Blacksburg, VA 24061

Abstract. We present a modular approach to realizing fine-grained adaptation of program behavior in a parallel environment. Using a compositional framework based on function call interception and manipulation, the adaptive logic to monitor internal program states and control the behavior of program modules is written and managed as a separate code, thus supporting centralized design of complex adaptation strategies for adapting to dynamic changes within an application. By 'catching' the functions that execute in synchronization across the parallel environment and inserting the adaptive logic operations at the intercepted control points, the proposed method provides a convenient way of synchronous adaptation without disturbing the parallel execution and communication structure already established in the original program. Applying our method to a CFD (computational fluid dynamics) simulation program to implement example adaptation scenarios, we demonstrate how effectively applications can change their behavior through fine-grained control.

1 Introduction

Implementing adaptive execution of an application in a distributed or parallel environment has been of much interest in recent years. The approaches to support program adaptation include: languages and compilers for specifying adaptation strategies [1,2,3] and runtime platforms or middleware for adaptive execution [4,5,6,7]. These efforts are primarily centered around resource management to achieve efficient utilization of the environment, such as adaptive load-balancing and scheduling of application tasks, to match resource constraints or dynamic operating conditions of the environment. Adaptation schemes are 'coarse-grained' in these approaches in that cooperating processes of a distributed application are each abstracted as a task and adaptation strategies are designed to reassign the tasks onto the resources or to reorganize the execution flow among them. The metrics to initiate adaptation are usually based on measured history or estimates of the application execution time, which is a function of the environment's operating conditions such as available resources (e.g., number of processors) or physical characteristics of the resources (e.g., network bandwidth).

Even with such support, however, adaptation schemes where functional behavior needs to be adjusted in response to internal changes to program state can be hard to design. Key challenges include the need for specifying adaptive parallel control points and monitoring state changes within the program, the need for executing parallel adaptation without disturbing the original execution flow, and the lack of support for centralizing adaptive logic operations in a separate module, thereby providing a compositional approach to dealing with the increased complexity of parallel adaptive applications. On the whole, coarse-grained approaches such as supported by runtime systems do not provide mechanisms to access fine-grained aspects of program state or to manipulate fine-grained behavior of the processes of a parallel application.

To address the issues, this paper presents a modular method for implementing fine-grained adaptive behavior with parallel programs using a function call interception (FCI) framework called *Invoke* [8]. Our work makes the following contributions:

– Factoring out the adaptation logic: A new code that implements the intended adaptive logic is written in a separate module and inserted by intercepting the functions of interest in a running application. This enables adaptation without code modification. We specifically target MPI programs written in the SPMD (Single Program, Multiple Data) style.
– Fine-grained control: Adaptation of program behavior such as simulation parameter adjustment or algorithm switching can be initiated in response to changes in internal computation states.
– Synchronous adaptation: By 'catching' global computation functions in the original program and plugging in new codes at the intercepted places, adaptive operations can be safely carried out without disturbing the parallel execution structure already established in the original program.

2 Compositional Approach for Parallel Adaptation

Invoke is a composition framework with a set of FCI APIs, through which every call to a function of interest is intercepted and program control is diverted to an associated *handler*, a piece of newly inserted code responsible for monitoring and modifying the target function's behavior. By specifying a target function to be manipulated by Invoke, we essentially define an adaptive control point over the original program, where newly developed modules can be introduced to maneuver the program toward the intended adaptive behavior. Thus, as Fig. 1 shows, composition through Invoke enables one to separately reason about application-specific adaptive strategies, factor them out in a centralized code, and plug in the adaptation code at control points to build an adaptive application.

2.1 Fine-Grained Program Adaptation

By defining adaptive control points at the interfaces of subprogram modules, the compositional approach conveniently achieves effective, fine-grained control over

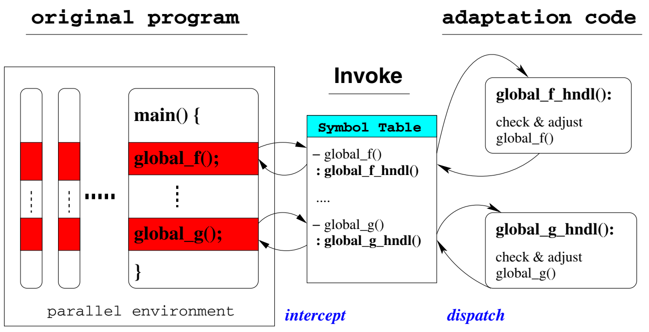


Fig. 1. Composition of an adaptive parallel application using Invoke

application behavior, where adaptation strategies can be designed to monitor and react to changes in internal program states. Global state variables can be accessed from the adaptivity code by declaring these variables as external. The Invoke framework also provides function parameter control APIs, which enables an extra level of flexibility in fine-grained adaptation. Function arguments are usually not exposed as globals in a program but still can hold important runtime program state for certain adaptation purposes. Through the parameter control APIs, dynamic program states that are communicated between modules can not only be accessed to check the computational progress, but also be manipulated to adjust the program's runtime behavior. We had previously presented such adaptation for sequential environments in [9] but here we focus on parallel execution environments.

2.2 Synchronous Parallel Adaptation

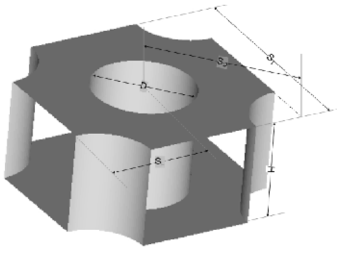
Implementing parallel adaptive behavior through the existing Invoke compositional framework requires adaptive logic operations to take place synchronously at clearly defined program control points that are shared across all the participating processes. This is important for implementing fine-grained adaptation strategies with SPMD programs where program behavior needs to change dynamically in response to changes in program state, because asynchronous adaptation in a parallel program can cause race conditions among the processes and make the entire computation invalid. For example, if one process changes a global simulation parameter or algorithm, and continues the computation, before another process makes the corresponding adaptation, the result may be inconsistent. Therefore, a synchronous adaptation mechanism is essential for implementing fine-grained adaptation in a parallel environment, where program behavior (typically in response to changes in internal program states) needs to adapt dynamically.

Synchronous adaptation can degrade performance if the adaptivity code involves extra global communication and synchronization. To mitigate the potential performance slowdown caused by adaptive global operations, we plug in the adaptivity code at global synchronization points that are already established in the original program, thus placing separate barriers (one from the original code and the other from the new adaptivity code) close together and making the combined overhead smaller. By having the adaptivity operations "piggyback" onto the existing communications that are executed synchronously across the parallel environment, monitoring and adjusting the program states can also be performed synchronously without explicitly using extra global operations.

3 Adaptive CFD Simulations

In this section, we apply our framework to the GenIDLEST CFD simulation code [10] to automatically adjust the simulation time step value and dynamically change the flow model. Written in Fortran 90 with MPI to simulate CFD problems, GenIDLEST solves the time-dependent incompressible Navier-Stokes and energy or temperature equations.

The simulated problem is a pin fin array geometry as shown in Fig. 2. Extended surfaces or fins have been used extensively to augment the heat/mass transfer from or to a surface primarily by increasing the transfer area and/or increasing the heat/mass transfer coefficient. Reducing the size and weight requirements of equipment necessitates the need for optimal designs of these systems, which in turn requires a detailed understanding of flow and heat transfer characteristics. The schematic and the geometric parameters of the pin fin array under consideration, along with the dimensions of interest, are listed in Fig. 2. The slenderness ratio is set to 1. For the GenIDLEST simulation, we divided the geometry into 16 block structures so that the maximum degree of parallelism is 16, where each block is assigned to one MPI process.



S_D	S_L	S_T	H	D
2.0	1.414214	2.828427	1	1

Fig. 2. Schematic and Geometric Parameters of Pin Fin Array under Consideration

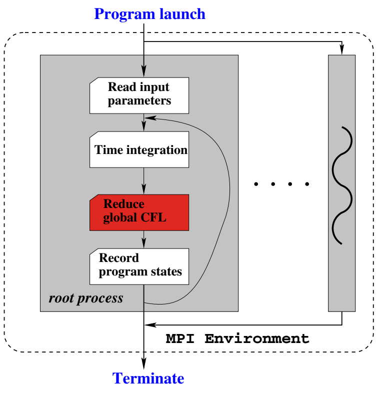


Fig. 3. GenIDLEST Execution Flow

3.1 Automatic Adjustment of Simulation Time Step

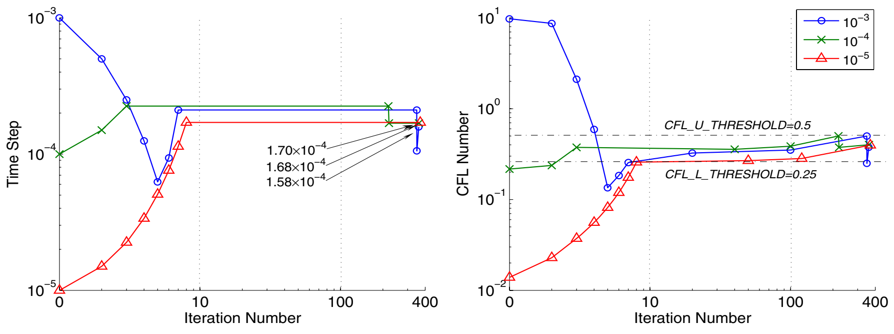
The stability of the simulation depends on the time step size used. Based on observed Courant-Friedrich-Levi (CFL) numbers one could discern if the simulation is proceeding towards convergence or is becoming unstable. Current practice of running GenIDLEST simulations records intermediate results at the end of a preset number of iterations onto the disk, thereby allowing the user to stop the execution and restart from the last known stable state when the user determines the running simulation is diverging. By plugging in a simple adaptivity module, the enhanced GenIDLEST simulation (requiring no modifications to the original GenIDLEST code) will incrementally adjust the time step value at runtime, allowing the computation to proceed in a stable manner.

Implementation: Fig. 3 shows the execution flow of the GenIDLEST simulation. At the end of every preset number of iterations, a local CFL number is calculated by each MPI process, and then the global CFL value is computed using a reduction operation (`mpi_allreduce`) across all the processes. This point is a good candidate for adaptivity code insertion, since by catching and imposing operations at this synchronization point, the newly inserted code can also be executed in synchronization, thereby avoiding dangerous race conditions among the processes. Furthermore, catching the global reduction call also makes it easy to monitor the global CFL number because its value is passed as the second parameter of the function. Invoke's parameter accessing APIs can be utilized to access this value. In the adaptive logic, we employed a simple multiplicative increase, multiplicative decrease algorithm with upper (`CFL_U_THRESHOLD`) and lower (`CFL_L_THRESHOLD`) threshold values for the CFL number, such that the time step is increased or decreased by a preset factor if the current CFL number becomes out of the bounds defined by the thresholds. Importantly, the entire adaptive logic operations are performed synchronously at the call sites of `mpi_allreduce` without involving any extra global operations, thereby achieving efficient parallel program adaptation. The implementation aspects of this and the following adaptation scenarios are summarized in Table 1.

Experimental Results: Fig. 4 shows the results of GenIDLEST enhanced with the constructed adaptivity module for the pin fin array simulation, with different initial values of time step ranging from 10^{-3} to 10^{-5}. `CFL_U_THRESHOLD` and `CFL_L_THRESHOLD` were set to 0.5 and 0.25, respectively. The graphs show how the CFL value changes as the time step parameter is controlled by the new

Table 1. Implementation Aspects of GenIDLEST Adaptation

	Change of Time Step	**Change of Flow Models**
PURPOSE	improve stability	enhance accuracy
TYPE OF SCHEME	automatic adjustment	user's dynamic decision
STATES TO MONITOR	CFL number communicated by `mpi_allreduce`	stream-wise velocity written to a log
CONTROL POINT	`mpi_allreduce` in `calc_cfl`	`calc_cfl` in time integration loop
ADAPTIVE LOGIC	adjust time step to confine CFL number within certain bounds	switch flow model (`i_les`) and activate turbulent data structures
COMMUNICATION	not necessary	broadcast of user's decision



(a) Time Step Change (b) CFL Number

Fig. 4. Automatic Adjustment of the Time Step Parameter

module, thereby maintaining the stability of the simulation. Interestingly, it also shows that the time step in all cases converge to somewhere around 1.7×10^{-4}, which might be the optimal value for the model, regardless of different starting values. Therefore, an adaptive logic based on a sophisticated CFD theory might be devised to find the optimal time step for more generalized problems through our composition method.

3.2 Runtime Change of Flow Models

The predicted heat transfer and flow characteristics depend on the selection of the appropriate flow model. A fundamental distinction is between laminar and turbulent flow models, and simulations of interest often require a switch from one to the other. This problem becomes acute when the Reynolds number is in the transition region between laminar and turbulent flows. Thus it becomes important to change the flow model from laminar to turbulent once instabilities arise in the flow field, for a simulation that is started assuming the flow is laminar.

Two Large Eddy Simulation (LES) turbulent models are considered in this study – Smagorinsky model (SM) and dynamic Smagorinsky model (DSM) [11]. The most commonly used model is the Smagorinsky model, where the eddy viscosity of the subgrid scales is obtained by assuming that the energy production and destruction are in equilibrium. The drawback of this model is that the model coefficient is kept constant, while in reality it should vary within the flow field depending on the local state of turbulence. The dynamic Smagorinsky model computes the model coefficient dynamically, which overcomes the deficiencies of the Smagorinsky model by locally calculating the eddy viscosity coefficient to reflect closely the state of the flow [11]. The advantage of the DSM model is that the need to specify the model coefficient is eliminated, making the model more self-contained, but with an additional computational expense of 10-15%.

Implementation: The simulated flow model in GenIDLEST is set initially by the user through an input specification parameter, namely `i_les`: 0 for laminar, 1 for Smagorinsky, and 2 for Dynamic Smagorinsky model. Hence, the program state needs to be accessed and changed at runtime via this variable. Importantly, the change should be made synchronously across all processes to maintain the consistency of the parallel computation. To this end, we plug in the adaptivity module at the call site of the CFL reduction function as shown in Fig. 3, because it is executed in synchronization across all the MPI processes, providing a safe place for carrying out adaptation operations without modifying the original code and disturbing the parallel execution flow already established in the original GenIDLEST. Specifically, the adaptivity code checks if the user wants to change the flow model, for which we make use of Unix signals (e.g., `SIGUSR1`) that can handle immediate, unanticipated user decisions to switch the flow model. These user-sent signals set a flag in the root process, which will pause accordingly with a simple user interface in the next iteration to accept the user's adaptation decisions, which in turn are broadcast to the other processes.

Experimental Results: The variation of the velocity in the direction of flow (stream-wise) is plotted in Fig. 5, showing the points in time when the flow models are switched from laminar to SM and then to DSM. The stream-wise velocity initially decreases, as the simulation proceeds towards the solution, which occurs till about 0.6 time units. After this simulation time, we see that the stream-wise velocity tends to vary with time, indicating the development of flow instabilities, and implying that the initial assumption of laminar flow is no longer valid. Thus the model is switched to SM at time 1.0. The drawback with the SM model, as mentioned earlier, is that the model coefficient is set to a constant value, but in reality the coefficient varies with the local state of turbulence, thus it becomes imperative to change the model from SM to DSM. This switch is done after a few hundred iterations (at time 1.4) to make sure that the switch from laminar to turbulent model does not introduce instabilities in the computation. The switch from laminar to turbulent flow model has a significant effect on the heat transfer. This is shown in Fig. 6(a) and 6(b), which show the variation of the Nusselt number at the channel walls, which is a measure of heat transfer at that location. The dotted line shows the region of interest, which is at the front of the pin in the line of fluid flow. The laminar flow model does not capture the

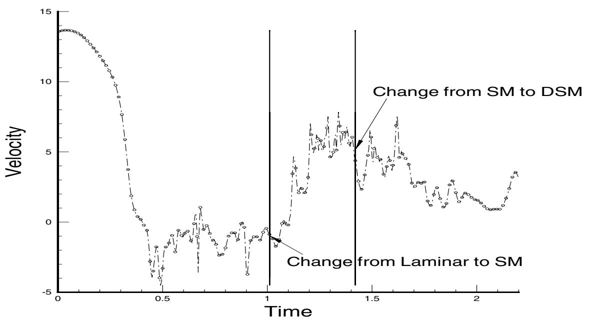
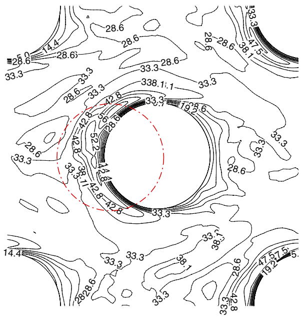
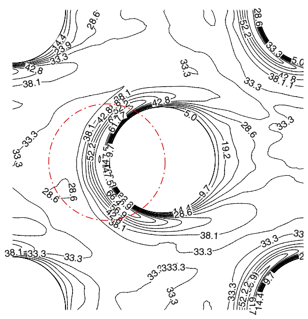


Fig. 5. Variation of Stream-Wise Velocity with Flow Model Change



(a) Laminar Model



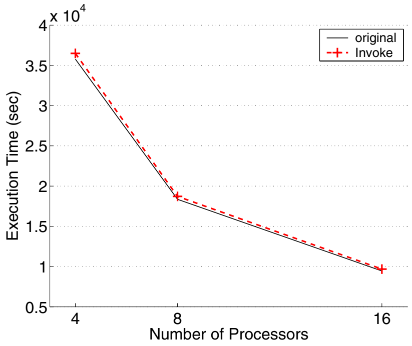
(b) Turbulent DSM Model

Fig. 6. Dynamic Flow Model Change from Laminar to Turbulent in a CFD Simulation

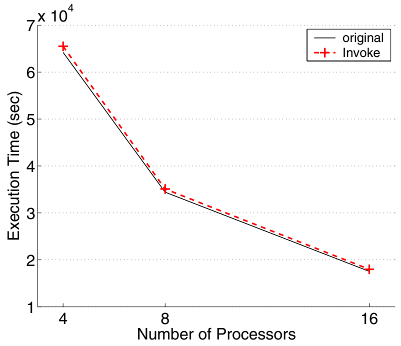
heat transfer effects at the front of the pin, predicting lower heat transfer rates at the pin front than the turbulent model, thus justifying the model switch from laminar to DSM. This switch shows the capabilities of the adaptive scheme, since to effect the switch without it would have meant stopping the current execution and then restarting the simulation after effecting the required change.

4 Adaptation Overhead

The runtime overhead of our adaptation method comes from catching the function calls at adaptive control points, which in itself does not involve any global operations that cause communication overhead. The catching overhead is measured at 0.10μs per call on average on an AMD Opteron 240 1.4GHz dual-processor machine with 1GB memory, which translates to 140 CPU cycles. Since the catching cost is fixed, the relative overhead depends on the number of interceptions and the entire execution profile of an application. That is, the overhead increases as the number of adaptive control points increases. Still, the catching cost is relatively insignificant if the application spends most of its time on executing other parts of the computation than at control points.



(a) Time Step Change (500 steps)



(b) Flow Model Change (1000 steps)

Fig. 7. Invoke Overhead with GenIDLEST Simulations

In the adaptive GenIDLEST simulations, control points are intercepted only once at the end of every preset number of iterations of the time integration loop, while most of the computing time is spent inside the loop. As a result, the catching overhead is negligible compared to the whole simulation profile. For example, Fig. 7 shows execution time of the GenIDLEST simulations where the Invoke framework is imposed at control points in the time step change and in the flow model change scenario, respectively, but with no adaptation operations. Across the 3 configurations with varying number of processors, the costs for catching 500 calls of `mpi_allreduce` during 500 time steps in the time step change example were measured to be less than 0.7% in all cases compared to the original GenIDLEST simulations (Fig. 7(a)). Similarly, the overhead is less than 0.95% for catching 1000 calls of `calc_cfl` during 1000 time steps in the flow model change example (Fig. 7(b)).

5 Related Work

In the language and compiler approaches for implementing program adaptation, our work is similar to Program Control Language (PCL) [2] in that centralized design of adaptation strategies can be specified at a high level for distributed programs. The expressive power of PCL comes from its underlying framework which offers a global representation of the distributed program as a graph of task nodes, the static task graph (STG), connected by edges indicating precedence relationships. Each adaptation primitive of PCL maps to a sequence of graph-changing operations on the STG of the target program. The Invoke framework provides more fine-grained control than PCL STGs by supporting monitoring and manipulating of state variables internal to a program.

In Grid and cluster computing, there is a large body of research work on runtime platforms for supporting program adaptation at the level of middleware or runtime platforms [4,5,6,7]. However, as their objective is to implement middleware support for adaptation between the application and the underlying execution layer, these efforts focus on resource management towards efficient utilization of the environment, such as load-balancing and scheduling of application tasks, where coarse-grained strategies based on resource constraints or external operating parameters are employed. In contrast, our work implements a parallel adaptation framework that can adjust fine-grained aspects of program state and behavior by monitoring dynamic progress of the computation itself.

Dynamic binary instrumentation tools such as DynInst [12] offer a modular, language-independent way of code modification, so that new code modules can be transparently combined with existing software. Since the accompanying overhead is significant while they perform code instrumentation at program runtime, they are usually developed for sophisticated programs analysis purposes [13] rather than as a tool to realize program behavior adaptation.

6 Conclusions

The proposed compositional framework offers a modular way of implementing fine-grained program adaptation in a parallel environment. By defining adaptive control points at the functions that execute in synchronization across the parallel environment, adaptive logic operations can safely be executed without interfering with the parallel execution structure of the original program. In future work, we intend to define 'adaptivity schemas' that abstract our recurring templates of adaptivity and that can be 'weaved' over an unmodified program, akin to aspect oriented programming. We also intend to explore more dynamic and less synchronous scenarios of parallel program adaptation.

References

1. Voss, M.J., Eigemann, R.: High-level Adaptive Program Optimization with ADAPT. In: PPoPP 2001: Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, pp. 93–102. ACM, New York (2001)
2. Ensink, B., Stanley, J., Adve, V.: Program Control Language: A Programming Language for Adaptive Distributed Applications. J. Parallel Distrib. Comput. 63(11), 1082–1104 (2003)
3. Du, W., Agrawal, G.: Language and Compiler Support for Adaptive Applications. In: SC 2004: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, Washington, DC, USA, pp. 29–29. IEEE Computer Society Press, Los Alamitos (2004)
4. Kennedy, K., Mazina, M., Mellor-Crummey, J.M., Cooper, K.D., Torczon, L., Berman, F., Chien, A.A., Dail, H., Sievert, O., Angulo, D., Foster, I.T., Aydt, R.A., Reed, D.A., Gannon, D., Johnsson, S.L., Kesselman, C., Dongarra, J., Vadhiyar, S.S., Wolski, R.: Toward a Framework for Preparing and Executing Adaptive Grid Programs. In: IPDPS 2002: Proceedings of the 16th International Parallel and Distributed Processing Symposium, Washington, DC, USA, pp. 171–175. IEEE Computer Society Press, Los Alamitos (2002)
5. Buaklee, D., Tracy, G.F., Vernon, M.K., Wright, S.J.: Near-Optimal Adaptive Control of a Large Grid Application. In: Proceedings of the 16th International Conference on Supercomputing, pp. 315–326. ACM Press, New York (2002)
6. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., Zagorodnov, D.: Adaptive Computing on the Grid using AppLeS. IEEE Transactions on Parallel and Distributed Systems 14(4), 369–382 (2003)
7. Janjic, V., Hammond, K., Yang, Y.: Using Application Information to Drive Adaptive Grid Middleware Scheduling Decisions. In: Proceedings of the 2nd Workshop on Middleware-Application Interaction, pp. 7–12. ACM, New York (2008)
8. Heffner, M.A.: A Runtime Framework for Adaptive Compositional Modeling. Master's thesis, Blacksburg, VA, USA (2004)
9. Kang, P., Cao, Y., Ramakrishnan, N., Ribbens, C.J., Varadarajan, S.: Modular Implementation of Adaptive Decisions in Stochastic Simulations. In: SAC 2009: Proceedings of the 24th Annual ACM Symposium on Applied Computing, pp. 995–1001 (March 2009)

10. Tafti, D.: GenIDLEST - A Scalable Parallel Computational Tool for Simulating Complex Turbulent Flows. In: Proceedings of the ASME Fluids Engineering Division (FED), vol. 256, ASME-IMECE (November 2001)
11. Germano, M., Piomelli, U., Moin, P., Cabot, W.H.: A Dynamic Subgrid-Scale Eddy Viscosity Model. Physics of Fluids A: Fluid Dynamics 3(7), 1760–1765 (1991)
12. Buck, B., Hollingsworth, J.K.: An API for Runtime Code Patching. Int. J. High Perform. Comput. Appl. 14(4), 317–329 (2000)
13. Schulz, M., Ahn, D., Bernat, A., de Supinski, B.R., Ko, S.Y., Lee, G., Rountree, B.: Scalable Dynamic Binary Instrumentation for Blue Gene/L. SIGARCH Comput. Archit. News 33(5), 9–14 (2005)

Evaluating Algorithms for Shared File Pointer Operations in MPI I/O

Ketan Kulkarni and Edgar Gabriel

Parallel Software Technologies Laboratory,
Department of Computer Science, University of Houston
{knkulkarni,gabriel}@cs.uh.edu

Abstract. MPI-I/O is a part of the MPI-2 specification defining file I/O operations for parallel MPI applications. Compared to regular POSIX style I/O functions, MPI I/O offers features like the distinction between individual file pointers on a per-process basis and a shared file pointer across a group of processes. The objective of this study is the evaluation of various algorithms of shared file pointer operations for MPI-I/O. We present three algorithms to provide shared file pointer operations on file systems that do not support file locking. The evaluation of the algorithms is carried out utilizing a parallel PVFS2 file system on an InfiniBand cluster and a local ext3 file system using a 8-core SMP.

1 Introduction

The MPI standards provide many features for developing parallel applications on distributed memory systems, such as point-to-point communication, collective operations, and derived data types. One of the important features introduced in MPI-2 is MPI-I/O [1]. The MPI I/O routines provide a portable interface for accessing data within a parallel application. A feature defined in MPI-I/O is the notion of a *shared file pointers*, which is a file pointer jointly maintained by a group of processes.

There are two common usage scenarios for shared file pointers in parallel applications. The first one involves generating event logs of a parallel application, e.g., to document the progress of each application process or to generate error logs. For these scenarios, users typically want to know the chronological order of events. Without shared file pointers, the parallel applications would be required to coordinate events with the other processes in a sequential manner (in the order in which these events occurred) [2]. Using a shared file pointer, the MPI library will automatically order the events, and ease the generation of parallel log files significantly. The second usage scenario uses a shared file pointer for assigning chunks of work to each processes in a parallel application. If a process has finished computing the work currently assigned to it, it could read the next chunk from the shared input file using a shared file pointer. Thus, no additional entity is required to manage the distribution of work across the processes.

Although there are/were some file systems that have support for shared file pointers such as VESTA [3], the majority of file systems available today do not provide a native support for these operations. Thus, the MPI library has to emulate shared file pointer operations internally. Today, the most wide-spread implementation of MPI I/O is ROMIO [4]. ROMIO relies on a hidden file which contains the shared file pointer for each MPI file. In order to modify the shared file pointer, a process has to lock the hidden file and thus avoid simultaneous access by multiple processes. However, many file systems have only limited or no support for file locking. As a result, shared file pointer operations often do not work with ROMIO [5]. Furthermore, since file locking is considered to be expensive, the performance of shared file pointer operations in ROMIO is often sub-optimal.

An alternative approach for implementing shared file pointer operations has been described in [6]. The solution proposed is to implement the shared file pointers using passive target one-sided communication operations, such as `MPI_Win_lock` and `MPI_Win_unlock`. A shared file pointer offset is stored in an MPI window. If a process wants to execute a read or write operation using the shared file pointer, it has to first *test* if any other process has acquired the shared file pointer. If this is not the case, it will access the shared file pointer from the root process. Else, it will wait for the signal from the process that currently has the shared file pointer. However, this approach is not available in a public release of ROMIO as of today, since it requires an implementation of one-sided operations, which can make progress outside of MPI routines, e.g. by using a progress thread [6]. In [7] Yu et.al. present an approach similar to one algorithm outlined in this paper by using the file-joining feature of Lustre to optimize the performance of collective write operations. In contrary to our algorithms however, there approach is bound to a specific file system and is not used to optimize shared file pointer operations.

In this paper we present three alternative algorithms for implementing shared file pointer operations on top of non-collaborative file systems. The first approach utilizes an additional process for maintaining the status of the shared file pointer. The second algorithm maintains a separate file for each process when writing using a shared file pointer. The third approach utilizes also an individual file on a per process basis, combines however the data of multiple MPI files into a single individual file. The remainder of the paper is organized as follows. In section 2 we describe the algorithms that have been developed, and document expected advantages, disadvantages and restrictions. Section 3 describes the test environment and the results of our evaluation over a parallel PVFS2 file system and a local ext3 file system. For the latter we also compare the performance of our algorithms to the ROMIO implementation of shared file pointers. Finally, section 4 summarizes the results of the paper and presents the ongoing work in this area.

2 Shared File Pointer Algorithms

The aim of designing new algorithms for shared file pointer operations is to avoid dependencies of the file systems on *file locks*, and to improve the performance of shared file pointer operations. In the following, we describe three algorithms to achieve these goals, namely using an additional process to maintain the shared file pointer, using an individual file per process and MPI file, and using a single file per process across all MPI files. The algorithms described in this section have been implemented as a standalone library using the profiling interface of MPI, and use individual MPI file pointer operations to implement the algorithms. In order to do not mix up the individual file pointers to an MPI file and the shared file pointer, each MPI File is opened twice providing two independent MPI handles.

2.1 Using an Additional Process

The main idea behind the first algorithm is to have a separate process that maintains the shared file pointer. This mechanism replaces the hidden file used by ROMIO maintaining the shared file pointer, thereby preventing the file locking problems mentioned previously. In the current implementation, the additional process is created by dynamically spawning a new process upon opening a file. However, in a long term we envision to utilize an already existing process, such as *mpirun* for this purpose, since the workload of the additional process is relatively low in realistic scenarios.

Upon opening a new MPI file, all processes of a process group defined by an intra-communicator collectively spawned the management process using `MPI_Comm_spawn`. `MPI_Comm_spawn` returns a new inter-communicator that consists of a local group containing the already existing processes and a new remote group containing the newly spawned process. This inter-communicator will be used for communication between the local and remote groups. The management process initializes and stores the shared file pointer. Whenever a participating application process wants to perform an I/O operation using the shared file pointer, it first requests the current position of the shared file pointer respectively the file offset from the management process. The management process puts all requests in a request-queue and sends the current value of the shared file pointer offset back to the requesting process, and increases the value by the number of bytes indicated in the request.

The collective version of the write operation (`MPI_File_write_ordered`) for shared file pointers is implemented in multiple steps. First, a temporary root process, typically the process with the rank zero in that communicator, collects from each process in the group the number of bytes that they would write to the main file using a gather operation. The temporary root process sends the request to the management process with the total number of bytes to be written as a part of this collective write operation, and receives the offset of the shared file pointer. It than calculates the individual offset for each process based on the shared file pointer offset and the number of bytes requested by each process, and sends it to each process individually. Note that an alternative implementation using `MPI_Scan` could be used to determine the local offset for each process. Each process then writes to the main file using the explicit offset collective blocking write file routine `MPI_File_write_at_all`. Thus, the collective notion of the operation is maintained. The collective read algorithm works in a similar fashion. This algorithm is able to implement the full set of functions defined in the MPI standard, and does not have any major restrictions form the usage perspective.

2.2 Using an Individual File Per Process and MPI File

This algorithm prevents file locking on the main file during collective and non-collective write operations using the shared file pointer by having each process write its data into an individual data file. Henceforth, the file that is opened and closed in the application would be referred to as the main file. At any collective file I/O operation the entire data from the individual files are merged into the main file based on the *metadata* information stored along with the application data. *Metadata* is the information about the data, which is being written into the data file, and contains the information shown in Table 1.

Table 1. Metadata Record

Record Id	ID indicating the write operation of this record
Timestamp	Time at which data is being written to the data file
Local position	Offset in the individual data file
Record length	Number of bytes written to the data file

In the following, we detail the required steps in this algorithm. Each process opens an individual *data file* and a *metadata file* during the `MPI_File_open` operation. During any individual write operation, each process writes the actual data into its individual data file instead of the main file. The corresponding Metadata is stored in the main memory of each process using a linked list. Once a limit to the number of nodes in the linked list is reached, for example, as indicated by the parameter *MAX_METADATA_RECORDS*, the linked list is written into the metadata file corresponding to that process. During the merging step, we first check for metadata records in the metadata file before reading the records from the linked list. The most important and time-consuming operation in this algorithm is that of merging independent data written by each of the process. Note that merging needs to involve all processes in the group. Hence, it can be done only during collective operations such as `MPI_File_write_ordered`, `MPI_File_write_ordered_begin`, and `MPI_File_close`.

The merging of data requires the following five steps:

1. All processes compute the total number of *metadata nodes* by combining the metadata nodes written by each of the processes using an `MPI_Allgather` operation.
2. Each process collects the timestamps and record lengths corresponding to all metadata nodes stored on each process using an `MPI_Allgatherv` operation.
3. Once each process receives all the metadata nodes containing the timestamps and record lengths, it sorts them based on the timestamps in an ascending order.

4. Each process assigns a global offset to the sorted list of the metadata nodes. The *global offset* corresponds to the position at which the data needs to be written into the main file.
5. Once each process has global offsets assigned to each metadata record, it reads the data from the local data file using the local file offset from the corresponding metadata node, and writes the data into the main file based on the global offset, which was computed in the previous step.

For the merging step we also envision the possibility to create the final output file in a separate post processing step. This would give applications the possibility to improve the performance of their write operations using a shared file pointer, and provide an automatic mechanism to merge the separate output files of the different processes after the execution of the application.

This implementation has three major restrictions:

– The individual file algorithm does not support read operations using the shared file pointers. The assumption within this prototype implementation is that the user indicates using an `MPI_Info` object that shared file pointers would not be used for read operations, respectively the MPI library should not choose this algorithm if the required guarantee is not given, but fall back to a potentially slower but safer approach.
– In order to be able to compare the timestamps of different processes, this algorithm requires a globally synchronized clock across the processes. While there are some machines providing a synchronized clock, this is clearly not the case on most clusters. However, there are algorithms known in the literature on how to synchronize clocks of different processes [8], which would allow to achieve a relatively good clock synchronization at run time. Nevertheless, this algorithm has the potential to introduce minor distortions in the order of entries between processes.
– This implementation can not support applications that use both individual and shared file pointers simultaneously. In our experience this is however not a real restriction, since we could not find any application using both type of file pointers at the same time.

An additional hint passed to the library using an Info object could indicate whether the merge operation shall be executed at runtime or in a post-processing step.

2.3 Using a Single Individual File Per Process

This algorithm offers a slight variation from the individual file per process algorithm. It maintains a single data and meta data file per process across all the files that have been opened by that process. Data pertaining to multiple files are written into single data and metadata files. The main reason behind this approach is to ease the burden on the metadata servers of the file systems, whose main limitation is the number of simultaneous I/O requests that can be handled.

The main difference between this algorithm and the one described in the previous subsection is in the handling of collective write operations. When a metadata node is written to the metadata file for an individual operation, each process can individually determine the according timestamp of this operation. However, for collective operations, the data needs to be written in the order of the ranks of the processes. To ensure this behavior, the timestamp corresponding to the process with rank zero is broadcasted to all processes in the communicator, and used for the corresponding collective entry. To identify the collective operations at file close, each process maintains a linked list with the timestamps corresponding to the collective operations.

This algorithm has the same restrictions as the 'individual file per process and MPI file algorithm' described above.

3 Performance Evaluation

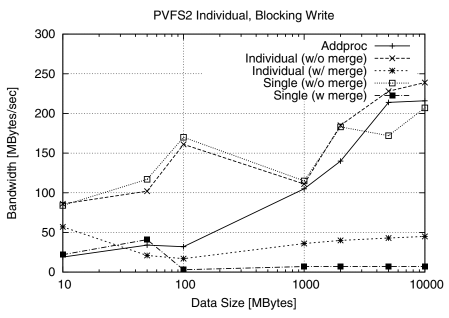
In the following, we present the performance of the algorithms described in the previous section. The three algorithms have been evaluated on two machines, namely, the *shark* cluster and *marvin*, an 8 core SMP server. The shark cluster consists of 24 compute nodes and one front end node. Each node consists of a dual-core 2.2GHz AMD Opteron processor with 2 GB of main memory. Nodes are connected by a 4xInfiniBand network. It has a parallel file system (PVFS2) mounted as '/pvfs2', which utilizes 22 hard drives, each hard drive is located on a separate compute node. The PVFS2 file system internally uses the Gigabit Ethernet network to communicate between the pvfs2-servers. Since two of the three algorithms presented in this paper only support write operations, our results section also focuses due to space constraints on the results achieved for various write tests.

Marvin is an eight processor shared memory system, each processor being a 2GHz single-core AMD Opteron processor. The machine has 8 GB of main memory and a RAID storage as its main file system using the Ext3 file system.

The results presented in this section are determined by using a simple benchmark program which writes data to a single file using either of the four shared file pointer write operations defined in MPI-2.

3.1 Results Using a PVFS2 File System

In the first test we constantly increase the total amount of data written by fixed number of processes. Tests have been performed for all four write operations using shared file pointers, namely, `MPI_File_write_shared`, `MPI_File_write_ordered`, `MPI_File_iwrite_shared`, `MPI_File_write_ordered_begin`. Due to space limitations, we show the results however only the blocking versions of these operations. Note that as explained in section [1], ROMIO does not support shared file pointer operations over PVFS2 [5]. In the following subsections, results obtained after comparing all three algorithms are listed. Every test has been repeated at least three times, and the minimum execution time has been used for each scenario.



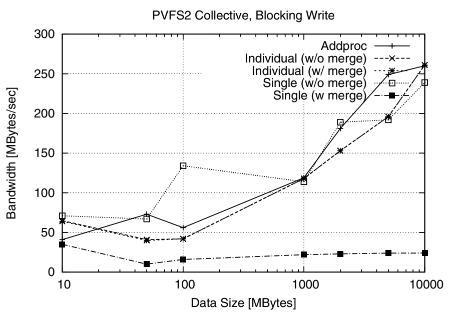


Fig. 1. Performance Comparison for Individual, Blocking write operation (left) and Collective Blockgin write-operations (right)

On PVFS2, the benchmark is run for each of the four write operations using shared file pointer. In each test case, data has been written from 10 MB to 10 GB among eight processes.

Figure 1 (left) gives the performance of individual and blocking write operation over PVFS2 file system. For the individual file and single file implementations, we show two separate lines for the execution time before merging (i.e., before calling `MPI_File_close`) and after merging. The reason for showing two lines is that these two algorithms are especially designed to minimize the time spent within the write operation itself, since the merging operation can potentially be performed in a post-processing step.

The individual file and single file implementations before merging of the data perform at par with the additional process implementation. All three implementations, with individual and single file performance compared before merging of data, show a relatively high bandwidth in the range of 140 MB/s to 240 MB/s, while writing 2-10 GB of data. The individual file algorithm achieves an overall bandwidth of around 50 MB/s after merging, while for the single file version bandwidth drops to around 10 MB/s including the merging step. The latter is the result of the large number of reduction and communication operations required in the merging step for that algorithm.

Figure 1 (right) gives the performance of the collective and blocking write operation over the PVFS2 file system. Note that the individual file implementation of the collective operations does not require merging, since the data is written directly into the main file for collective operations. Hence, only the single file implementation performance before the merging is compared with that of the other two implementations. All the three implementations perform similarly, only the performance of the single file method after the merging deviates significantly. The performance improves as the amount of data being written increases.

Performance of the three algorithms has also been evaluated using by writing a fixed amount of data, and varying the number of processes writing the data. Accordingly, on PVFS2 2GB of data is being written by varying the number of processes from 2 to 32. The left part of Fig. 2 shows that the additional process algorithm mostly outperforms the other two implementations. Although

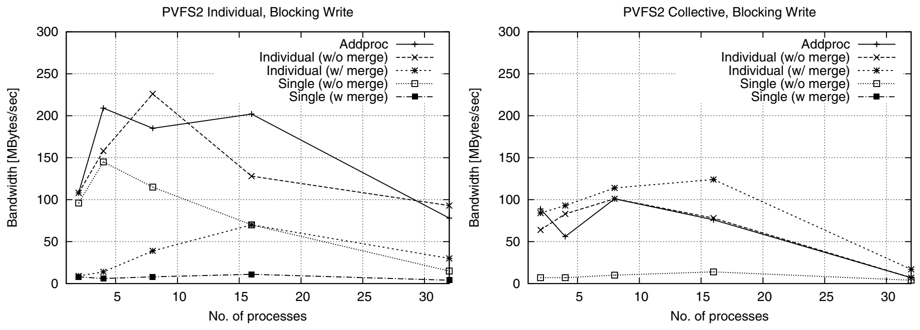


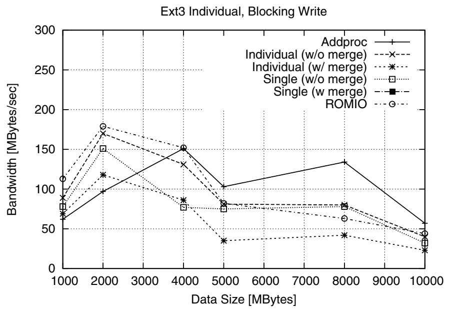
Fig. 2. Performance Comparison for Individual, Blocking write operation (left) and Collective Blockgin write-operations (right) for varying numbers of processors

it performs reasonably well up to 16 processes, the performance drops sharply for 32 processes. This performance drop can be due to either of the two reasons: the management process could be overwhelmed by the increasing number of (small) requests with increasing number of processes, or the file system or the hard drive could get congested with increasing number of requests.

Which of these two contributes stronger to the performance degradation observed for 32 processes can be answered by looking at the numbers of the collective write operations in the right part of Fig. 2. This graph shows the same behavior for the additional process implementation. However, as collective operations in this model only require one request to the management process per collective function call, but generate the same number of I/O requests to the hard drive and the file system, we are confident that the performance drop observed in the previous measurements is not due to a congestion at the additional process managing the shared file pointer.

3.2 Results Using an EXT3 File System

In this subsection, the benchmark is run using all four write operations of the MPI specification for shared file pointer operations using all the three algorithms described earlier on a local file system using Ext3. The test machine utilized consists of eight single core AMD Opteron processors. Since the Ext3 file system supports file locks, the performance of ROMIO over Ext3 could be compared with our three implementations. In each case, data from 1 GB to 10 GB is written among eight processes. The left part of Fig. 3 gives the performance of the individual and blocking write routine over Ext3 file system, while the right part shows the performance of the collective write routine. ROMIO performs better than the three new algorithms for smaller amounts of data. However, as the amount of data written increases, the performance of ROMIO decreases. The additional process implementation performance in these scenarios is better than ROMIO. There could be various reasons behind the additional process algorithm not being able to completely outperform ROMIO, such as



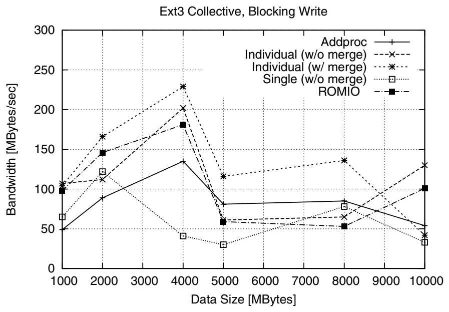


Fig. 3. Performance Comparison for Individual, Blocking write operation (left) and Collective Blockgin write-operations (right) on the ext3 file system

– All the evaluation has been done using eight processes. Marvin is a shared memory system which has eight processors. In the case of the additional process algorithm, when an additional process is spawned (the ninth in this case), the performance might decrease as nine processes run on eight processors.
– Secondly, when the dynamic process management feature is used, Open MPI restricts the communication among the original process group and the newly spawned processes from using shared memory communication. Instead, it uses the next available communication protocol, which is TCP/IP in case of Marvin. Hence, slow communication between the local and the remote groups of processes could be a factor affecting the performance of the additional process algorithm.
– Thirdly, in case of ROMIO, since the hidden file is small, it might be cached by the file system or the RAID controller. Hence, processes accessing the hidden file might not touch the hard drive, but access the hidden file from the cache to read and update the shared file pointer offset value.

4 Summary

This study aims at designing portable and optimized algorithms to implement shared file pointer operations of the MPI I/O specification. We have developed, implemented and evaluated three new algorithms for I/O operations using shared file pointers. Our evaluation shows, that the algorithms often outperform the current version in ROMIO, although there are notable exceptions. However, the algorithms do represent alternative approaches which could be used in case the file system does not support file locking. The 'additional process' algorithm is most general of the three approaches, and we plan to further extend it by overcoming the restrictions discussed in section 3, most notably having the management process being executed in an already existing process, such as `mpirun`. Furthermore, as of today, we have not yet evaluated the difference between the single file and the individual file approaches in case of multiple MPI files, and the impact on the Metadata server of the file system. The ultimate goal is to provide a collection of algorithms and give the end-user the possibility to select between those algorithms using appropriate hints.

References

1. Message Passing Interface Forum: MPI-2: Extensions to the Message Passing Interface (1997), http://www.mpi-forum.org
2. May, J.: Parallel I/O for High Performance Computing. Morgan Kaufmann Publishers, San Francisco (2003)
3. Corbett, P.F., Feitelson, D.G.: Design and implementation of the Vesta parallel file system. In: Proceedings of the Scalable High-Performance Computing Conference, pp. 63–70 (1994)
4. Thakur, R., Gropp, W., Lusk, E.: Data Sieving and Collective I/O in ROMIO. In: FRONTIERS 1999: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation, Washington, DC, USA, p. 182. IEEE Computer Society, Los Alamitos (1999)
5. Thakur, R., Ross, R., Lusk, E., Gropp, W.: Users Guide for ROMIO: A High Performance, Portable MPI-IO Implementation. Argonne National Laboratory (2004)
6. Latham, R., Ross, R., Thakur, R.: Implementing MPI-IO Atomic Mode and Shared File Pointers Using MPI One-Sided Communication. Int. J. High Perform. Comput. Appl. 21(2), 132–143 (2007)
7. Yu, W., Vetter, J., Canon, R.S., Jiang, S.: Exploiting lustre file joining for effective collective io. In: CCGRID 2007: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, pp. 267–274. IEEE Computer Society Press, Los Alamitos (2007)
8. Becker, D., Rabenseifner, R., Wolf, F.: Timestamp Synchronization for Event Traces of Large-Scale Message-Passing Applications. In: Proceedings of EuroPVM/MPI. LNCS, pp. 315–325. Springer, Heidelberg (2007)

Computer Networks

Load Balancing Scheme Based on Real Time Traffic in Wibro

Wongil Park[1] and Hyoungjin Kim[2,*]

[1] Korea Institute of Construction Technology, Korea
prudent_woman@yahoo.co.kr
[2] Division of Applied Systems Engineering, Chonbuk National University, Korea
kim@chonbuk.ac.kr

Abstract. The WiBro operates at the 2.3GHz broadband and the communication infrastructure is a cellular system. The WiBro is based on IEEE 802.16e standard and it is designed to maintain connectivity on mobile environment at a speed of up to 60 km/h. ACR(Access Control Router) manages several RAS(Radio Access Station). When mobile node moves to another domain from the present domain, which is managed by different ACR, MN sends Binding Update to HA (Home Agent) or CN (Correspondent Node). However ACR may be a single point of performance bottleneck because the ACR should not only handle signaling traffics but also process data tunneling traffic for all MNs registered in its domain. In this paper, we propose ACR load balancing method by priority queue. Quantitative results of the performance analysis show that our proposal has superior performance.

Keywords: Wibro, ACR, real time data, non real time data, queue.

1 Introduction

In February 2002, Korean government allocated 100 MHz bandwidth of 2.3GHz spectrum band for wibro(Wireless Broadband)system. WiBro allows subscribers to use high-speed Internet more economically and more widely, even when moving at the speed of about 60km per hour [1]. As illustrated in figure 1, the wibro system consists of PSS(Portable Subscriber Station), RAS(Radio Access Station), ACR (Access Control Router) and IP based backbone networks[2,3].

Figure 1(a) and (b) show two cases of WiBro structure. Figure 1(a) has two subnets and each ACR manages several RAS within a subnet.

In this paper, we only consider the situation when a mobile node moves to another subnet, which is managed by different ACR.

All mobile nodes included in an ACR are to have mobility. Therefore, if MN increases to control by ACR, ACR grow BU(Binding Update). As a result, ACR cannot respond quickly to real time data rather than non real time data.

The rest of the paper is organized as follows. Section 2 presents the previous works about mobile multimedia process in WiBro. Section 3 proposes our method. Section 4

[*] Corresponding author.

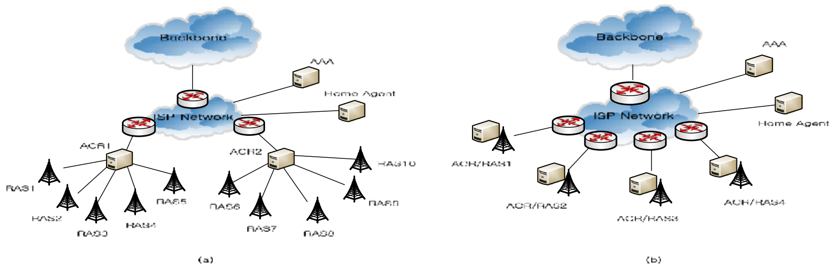


Fig. 1. A structure of WiBro

evaluates the performance and analyzes numerical results. Finally, we conclude this paper in section 6.

2 Related Work

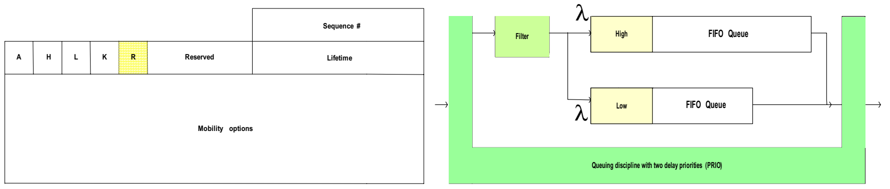
Shim, Kim and Ra [4] proposed a handover scheme for an efficient and reliable multicast routing over WiBro service. In order to dynamically manage each multicast flow and minimize the frequency of the multicast group join while guaranteeing the optimal path, they adopted FA based multicast routing scheme that is based on hierarchical architecture among ACR and RASs.

This paper has attempted to justify that sleep mode can be effective even in a mobile environment by adopting the optimized initiation process [5] and Wu and Kim proposed an efficient direction and speeds based handover connection control schemes for increasing the utilization of channels and reduce a probability of new connection blocking rate [6].

They proposed an efficient IPv6 based fast handover scheme for seamless inter-domain mobility support over WiBro networks considering cross-layer approach [7]. The FMIPv6 protocol has problem to be used with WiBro system, owing to difficulty in utilizing the layer 2 handover information. So, Shim, Kim and Lee proposed mechanism that can provide effective fast handover in IPv6 based WiBro system [8] and W. Lee et al proposed an adaptive vertical handoff decision scheme called Ubi-Comm which is an improved handover decision algorithm that avoids the ping-pong effect [9]. In [10], they proposed algorithm utilizes the user based scheduling to relieve the MAP(Mobility Anchor Point) overhead problem and to modify the normal proportional fair scheduling algorithm to guarantee user based QoS.

3 Proposed Method

As mentioned above, ACR may be a single point performance bottleneck because the ACR should not only handle signaling traffic but also process data tunneling traffic for all MNs registered to the ACR domain. There are many works performed on multimedia data process, but not on BU (Binding Update) process in ACR. As such, we propose new method to operate BU in ACR's waiting queue in order to update new location of MN rapidly.



(a) New binding update message

(b) Binding update processing with priority queues in ACR

Fig. 2. The process of proposed method

We show the priority process of ACR in (a) of Fig 2. For an instance, if ACR has many MN of mobility, ACR increases the overall BU process. Compared to the existing method, which has only one queue, waiting time is increased. Besides, existing method employs FIFO. However, if our proposed method is applied, there would be two queues and the general performance is improved when the waiting time decreases. Fig. 3 shows proposed system model.

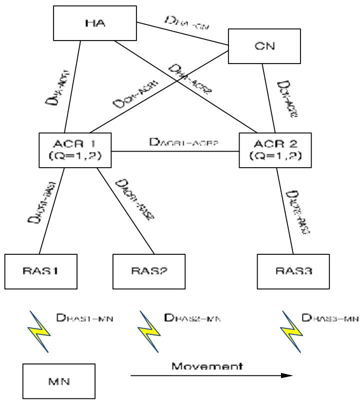


Fig. 3. The picture is proposed method

4 Numerical Analysis

4.1 Mobility Model

We assumed that there is hexagonal cellular network architecture, as shown in Fig. 4. Each ACR domain consists of the different number of range rings, D. Rings of cells surround each cell as illustrated in Fig. 4 [11]. Each ring d (d>=0) is composed of 6d cells. The innermost cell "0" is called the center cell. The number of cells N (D) is calculated using the following equation:

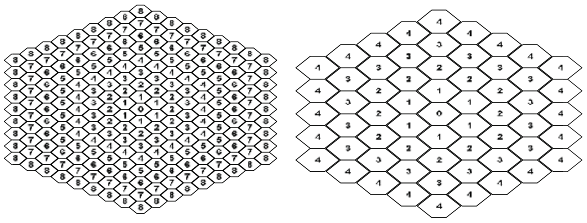


Fig. 4. Hexagonal Cellular Network Architecture (D=8, D=4)

$$N(D) = 1 + 6 \cdot \sum_{d=1}^{D} d = 1 + 3 \cdot D \cdot (D+1) \tag{1}$$

Hexagonal cellular network architecture (D=8) of Fig. 4 shows the proposed hexagonal cellular network architecture of traffic characteristics.

The two-dimensional model used in Markov chain model with respect to the user mobility model is considered [12]. In this model, the next position of an MN is equal to the previous position in addition to a random variable whose value is drawn independently from an arbitrary distribution. Besides, an MN moves to another cell area with a probability of 1-q and it remains in the current cell with probability, q. Given an MN located in a cell of ring d (d>0), the probability that a movement will cause an increase (p+(d)) or decrease(p-(d)) in distance from the center cell is given by

$$P^{+}(d) = \frac{2d+1}{6d} \quad and \quad P^{-}(d) = \frac{2d-1}{6d} \tag{2}$$

The state k of a Markov chain is defined as the distance between the current cell of the MN and the center cell. This state is equivalent to the index of a ring in which the MN is located. Therefore, the MN is said to be in state k if it is currently residing in ring d. The transition probabilities $\alpha_{d,d+1}$ and $\beta_{d,d-1}$ represent the probabilities of the distance of the MN from the center cell increasing or decreasing, respectively. They are given as follow, where q is the probability that an MN stays in the current cell:

$$\alpha_{d,d+1} = \begin{cases} (1-q) & if \quad d = 0 \\ (1-q)\,p^{+}(d) & if \quad 1 \le d \le D \end{cases} \tag{3}$$

$$\beta_{d,d-1} = (1\text{-}q)\ p^{-}(d) \qquad if \quad 1 \le d \le D \tag{4}$$

We denote $p_{d,D}$ as the steady-state probability of state d within a ACR domain consisting of D range rings. As stated in Equation (3) and Equation (4), $P_{d,D}$ can be expressed in the form of the steady state probability $P_{0,D}$ as shown below:

$$P_{d,D} = P_{0,d} \prod_{i=0}^{d-1} \frac{\alpha_{i,i+1}}{\beta_{i+1,i}} \qquad \text{for } 1 \le d \le D \tag{5}$$

With the requirement $\sum_{d=0}^{D} P_{d,D} = 1$, $P_{d,D}$ can be expressed by

$$P_{0,D} = \frac{1}{1 + \sum_{d=1}^{D} \prod_{i=0}^{d-1} \frac{\alpha_{i,i+1}}{\beta_{i+1,i}}} \tag{6}$$

where $\alpha_{d,d+1}$ and $\beta_{d,d-1}$ are obtained from Equation(3) and Equation(4)

4.2 Cost Functions

The total cost, consisting of location update cost and paging cost, should be considered when analyzing the performance of wireless/mobile networks. The total cost is divided into location update cost and packet delivery cost. In our proposed scheme, we divide total cost into new location update and packet delivery cost. $C_{location}$, $C_{new-location}$, and C_{packet} denote the location update cost, new location update and the packet delivery cost, respectively. As such, the total cost of MIPv6 (C_{total}) and the proposed scheme ($C_{new-total}$) can be obtained as follows:

$$C_{total} = C_{location} + C_{packet} \tag{7}$$

$$C_{new-total-high} = C_{NEW-location} + C_{packet-high} \tag{8}$$

$$C_{new-total-low} = C_{NEW-location} + C_{packet-low} \tag{9}$$

4.2.1 Location Update Cost

MN registers its CoA with the CNs and the HA, when a MN moves into a new domain.

$$\begin{aligned} C_g &= 2 \cdot (k \cdot D_{RAS\{1,2,3\}-MN} + \tau \cdot (D_{HA-ACR\{1,2\}} + D_{ACR\{1,2\}-RAS\{1,2,3\}})) \\ &+ 2 \cdot N_{CN} \cdot (k \cdot D_{RAS\{1,2,3\}-MN} + \tau \cdot (D_{HA-ACR\{1,2\}} + D_{CN-ACR\{1,2\}})) \\ &+ (PC_{HA} + W_Q) + N_{CN} \cdot (PC_{CN} + W_Q) + (PC_{ACR} + W_Q) \end{aligned} \tag{10}$$

$$C_l = 2 \cdot (k \cdot D_{RAS\{1,2,3\}-MN} + \tau \cdot (D_{ACR\{1,2\}-RAS\{1,2,3\}}) + PC_{ACR} + W_Q \tag{11}$$

$$\begin{aligned} C_{NEW-g} &= 2 \cdot (k \cdot D_{RAS\{1,2,3\}-MN} + \tau \cdot (D_{HA-ACR\{1,2\}} + D_{ACR\{1,2\}-RAS\{1,2,3\}})) \\ &+ 2 \cdot N_{CN} \cdot (k \cdot D_{RAS\{1,2,3\}-MN} + \tau \cdot (D_{HA-ACR\{1,2\}} + D_{CN-ACR\{1,2\}})) \\ &+ (PC_{HA} + W_Q) + N_{CN} \cdot (PC_{CN} + W_Q) + (PC_{ACR} + W_Q^1) \end{aligned} \tag{12}$$

$$C_{NEW-l} = 2 \cdot (k \cdot D_{RAS\{1,2,3\}-MN} + \tau \cdot (D_{ACR\{1,2\}-RAS\{1,2,3\}}) + PC_{ACR} + W_Q^2 \tag{13}$$

where τ and k indicate the unit transmission costs in a wired and a wireless link, respectively. PC_{HA} and PC_{CN} denotes the processing costs for binding update procedures at the HA and the CN, respectively. Given $D_{HA\text{-}ACR\{1,2,3\}}$, $D_{ACR\{1,2,3\}\text{-}RAS}$, $D_{RAS\text{-}MN}$ and $D_{CN\text{-}ACR\{1,2,3\}}$ as the hop distance between nodes, N_{CN} represents the number of CNs that is communicating with the MN.

In terms of the random walk mobility model, the probability that a MN performs a global binding update is as follows:

$$P_{D,D} \cdot \alpha_{D,D+1} \tag{14}$$

Specifically, if a MN is located in range ring D, the boundary ring of a ACR domain is composed of D range rings, and performs a movement from range ring D to range ring D+1.

$$C_{location} = \frac{P_{D,D} \cdot \alpha_{D,D+1} \cdot C_g + (1 - P_{D,D} \cdot \alpha_{D,D+1}) \cdot C_l}{T} \tag{15}$$

$$C_{NEW-location} = \frac{P_{D,D} \cdot \alpha_{D,D+1} \cdot C_{NEW-g} + (1 - P_{D,D} \cdot \alpha_{D,D+1}) \cdot C_{NEW-l}}{T} \tag{16}$$

where T is the average cell residence time.

4.2.2 Packet Delivery Cost

The packet delivery cost, C_{PACKET} , in WiBro can then be obtained as follows:

$$C_{PACKET} = C_{ACR} + C_{CN-MN} \tag{17}$$

$$C_{PACKET-HIGH} = C_{ACR-HIGH} + C_{CN-MN} \tag{18}$$

$$C_{PACKET-LOW} = C_{ACR-LOW} + C_{CN-MN} \tag{19}$$

C_{ACR} in Equation(17) indicates the processing cost for packet delivery at the ACR, while C_{CN-MN} in figure 3 denotes the packet transmission cost from the CN to the MN.

In WiBro, the processing cost at the ACR is divided into the lookup cost (C_{lookup}), the routing cost ($C_{routing}$) and the waiting time (C_{wait}) in queue. The lookup cost is proportional to the size of the mapping table, whereas the size of the mapping table is proportional to the number of MNs located in the coverage of a domain [13]. On the other hand, the routing cost is proportional to the logarithm of the number of ARs within a particular domain [14]. The waiting time denotes the priority [15]. Therefore, the processing cost at the ACR can be expressed as Equation (25). In Equation (25), λ denotes the session arrival rate while S denotes the average session size in the unit of packet. α and β are the weighting factors, and N_{MN} shows the total number of users located in a domain.

The M/G/1 model assumes (i) Poisson arrivals at rate λ; (ii) a general service distribution; and (iii) a single server. This follows since there is only a single server. Considering expectations of both sides of customer's wait in queue yields

$$W_Q = \text{Average work as seen by an arrival.}$$

However, owing to Poisson arrivals, the average work as seen by an arrival will equal V, the time average work in the system. Hence, for the model M/G/1

$$W_Q = V \tag{20}$$

The proceeding in conjunction with the identity

$$V = \lambda E[S]W_Q + \frac{\lambda E[S^2]}{2} \tag{21}$$

yields the so-called Pollaczek-Khintchine formula:

$$W_Q = \frac{\lambda E[S^2]}{2(1-\lambda E[S])} \tag{22}$$

Priority queuing systems are ones in which customers are classified into types and then given service priority according to their type. Consider the situation where there are two types of customers, who arrive according to independent Poisson processes with respective rates λ_1 and λ_2. Let W_Q^i denote the average wait in queue of a type i customer, i = 1, 2. Our objective is to compute W_Q^i.

The work in the system is exactly the same as the work when there is no priority rule but rather a first-come, first-served (called FIFO) ordering. However, under FIFO the preceding model is just M/G/1 with

$$\lambda = \lambda_1 + \lambda_2 \tag{23}$$

which follows since the combination of two independent Poisson processes is itself a Poisson process whose rate is the sum of the rates of the component processes.

$$W_Q^1 = \frac{\lambda_1 E[S_1^2] + \lambda_2 E[S_2^2]}{2(1-\lambda_1 E[S_1])} \tag{24}$$

$$W_Q^2 = \frac{\lambda_1 E[S_1^2] + \lambda_2 E[S_2^2]}{2(1-\lambda_1 E[S_1] - \lambda_2 E[S_2])(1-\lambda_1 E[S_1])} \tag{25}$$

This paper assumes that the average number of users located in the coverage of an AR is K. Therefore, the total number of users can be obtained using Equation (26).

$$N_{MN} = N_{AR} \times k \tag{26}$$

$$\begin{aligned} C_{ACR} &= \lambda \cdot \overline{S} \cdot (C_{lookup} + C_{routing} + C_{wait}) \\ &= \lambda \cdot \overline{S} \cdot (\alpha N_{MN} + \beta \log(N_{AR}) + W_Q) \end{aligned} \tag{27}$$

$$\begin{aligned} C_{ACR-HIGH} &= \lambda \cdot \overline{S} \cdot (C_{lookup} + C_{routing} + C_{wait}) \\ &= \lambda \cdot \overline{S} \cdot (\alpha N_{MN} + \beta \log(N_{AR}) + W_Q^1) \end{aligned} \tag{28}$$

$$\begin{aligned} C_{ACR-LOW} &= \lambda \cdot \overline{S} \cdot (C_{lookup} + C_{routing} + C_{wait}) \\ &= \lambda \cdot \overline{S} \cdot (\alpha N_{MN} + \beta \log(N_{AR}) + W_Q^2) \end{aligned} \tag{29}$$

Since WiBro supports the route optimization, the transmission cost in WiBro can be obtained using Equation(28). As mentioned before, τ and k denote the unit transmission costs in a wired and a wireless link, respectively.

$$\begin{aligned} C_{CN-MN} &= \tau \cdot \lambda \cdot ((S-1) \cdot (D_{CN-ACR\{1,2\}} \cdot D_{ACR\{1,2\}-RAS\{1,2,3\}}) \\ &+ (D_{HA-CN} + D_{CN-ACR\{1,2\}} \cdot D_{ACR\{1,2\}-RAS\{1,2,3\}})) + k \cdot \lambda \cdot S \end{aligned} \tag{30}$$

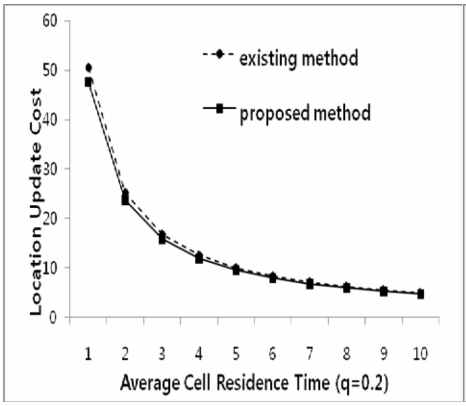
5 Numerical Results

In this section, we provide some numerical evaluation to demonstrate the performance of proposed scheme as compared with existing method. The parameter values for the analysis are referred from [16], [17] and [18]. They are shown in table 1.

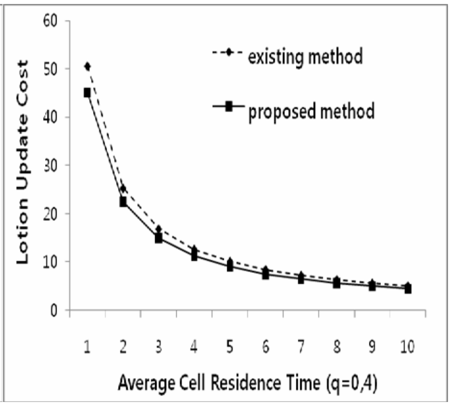
Table 1. Numerical simulation parameter for performance analysis

Parameter	α	β	τ	κ	DHA-ACR{1,2}	DCN-ACR{1,2}	DRAS{1,2,3}-MN	DACR{1,2}-RAS{1,2,3}
value	0.1	0.2	1	2	8	6	1	2
Parameter	DHA-CN	N_{CN}	PC_{HA}	PC_{CN}	PCACR	DACR1-ACR2	λ_1	λ_2
value	6	2	24	6	10	1	0.1	0.2

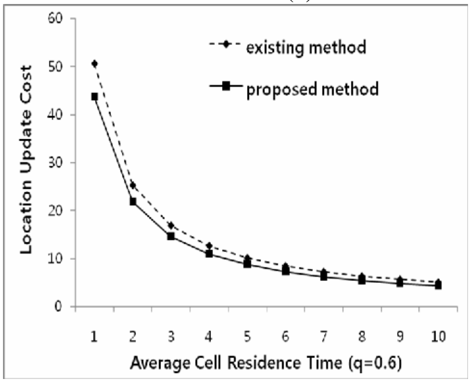
Fig. 5 shows the variation in the location update cost as the average cell residence time is changed in the random-walk model. In a comparison of our proposed scheme with the existing method, our proposed scheme reduces the location update cost from 16% to 8% approximately.



(a) $q = 0.2, \lambda = 0.8$



(b) $q = 0.4, \lambda = 0.6$



(c) $q = 0.6, \lambda = 0.4$

Fig. 5. Location update cost as function of average cell residence time of MN

6 Conclusion

The proliferation of the internet services with provision of a variety of multimedia content, as well as demand for portable internet service via various mobile terminals such as notebook PC, PDA, and cellular phone is increasing rapidly. However, current roaming procedures are very slow and unstable when we use real-time applications. In this paper, our work focuses on mobility based on the WiBro. We distinguish BU priority in ACR. The performance analysis and the numerical results presented in this paper show that our proposal has superior performance compared to the existing method. The proposed scheme reduces the location update cost by more than 15% approximately.

References

1. Park, Y., Adachi, F.: Enhanced Radio Access Technologies for Next Generation Mobile Communication (2007)

2. Specifications for 2.3GHz band Portable Internet Service-Physical & Medium Access Control layer, TTA (June 2005)
3. High-speed Portable Internet Handover Specification, ETRI (2003)
4. Shim, M., Kim, H., Ra, I.: A Handover Mechanism for Reliable Multicast Service over WiBro Networks. In: Proc. of the 9th IEEE International Conference on Advanced Communication Technology, February 2007, pp. 1081–1085 (2007)
5. Park, J., Kim, B., Hwang, I.: Handover-Specific Optimization for IEEE 802.16e Sleep Mode. In: Cham, T.-J., Cai, J., Dorai, C., Rajan, D., Chua, T.-S., Chia, L.-T. (eds.) MMM 2007. LNCS, vol. 4352, pp. 560–567. Springer, Heidelberg (2007)
6. Wu, M., Kim, C.: A Calculation Method for Direction Based Handover. Rate in Cell Based Mobile Networks. 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. 3983, pp. 818–827. Springer, Heidelberg (2006)
7. Kim, M., Ra, I., Yoo, J., Kim, D., Kim, H.: Cross-layer based Fast Handover Mechanism for Seamless Macro-mobility Support in WiBro Networks. In: ICOIN 2008, Busan, Korea (January 2008)
8. Shim, M., Kim, H., Lee, S.: A Fast handover Mechanism For IPv6 Based WiBro System. In: ICACT 2006, February 20-22 (2006)
9. Lee, W., Kim, E., Yu, J., Lee, D., Choi, J., Kim, J., Shin, C.K.: UbiComm: An Adaptive Vertical Handoff Decision Scheme for Heterogeneous Wireless Networks. In: Youn, H.Y., Kim, M., Morikawa, H. (eds.) UCS 2006. LNCS, vol. 4239, pp. 344–356. Springer, Heidelberg (2006)
10. Kim, J., Kim, E., Kim, K.: A new efficient BS scheduler and scheduling algorithm in WiBro systems. In: The 8th International Conference on Advanced Communication Technology, vol. 3, pp. 20–22 (February 2006)
11. Ho, J.S.M., Akyildiz, I.F.: Mobile user location update and paging under delay constrainsts. ACM-Baltzer J. Wireless Networks 1, 413–425 (December 1995)
12. Akyildiz, I.F., Wang, W.: A dynamic location management scheme for next-generation multitier PCS systems. IEEE Trans. Wireless Commun. 1(1), 178–189 (2002)
13. Pack, S., Choi, Y.: A Study on performance of hierarchical mobile IPv6 in IP-based cellular networks. IEICE Transactions on Communications E87-B(3), 462–469 (2004)
14. Pack, S., Lee, B., Choi, Y.: Load Control Scheme at Local Mobility Agent in Mobile IPv6 Networks. In: WWC 2004 (May 2004)
15. Ross, S.M.: Introduction to PROBABILITY MODELS (1997)
16. Woo, M.: Performance analysis of mobile IP regional registration. In: IEICE Trans. Commun., vol. E86-B(2), pp. 472–478 (February 2003)
17. Zhang, X., Castellanos, J.G., Capbell, A.T.: P-MIP: Paging extensions for mobile IP. ACM Mobile Networks and Applications 7(2), 127–141 (2002)
18. Park, W., Kim, B.: Fast BU process method for real time multimedia traffic in MIPv6. 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. 330–339. Springer, Heidelberg (2006)

Hybrid Retrieval Mechanisms in Vehicle-Based P2P Networks*

Quanqing Xu[1], Heng Tao Shen[2], Zaiben Chen[2], Bin Cui[1], Xiaofang Zhou[2], and Yafei Dai[1]

[1] State Key Lab for Adv Opt Commun Syst & Networks, Peking University, 100871 Beijing, China
{xqq,dyf}@net.pku.edu.cn, bin.cui@pku.edu.cn

[2] School of ITEE, The University of Queensland, Brisbane QLD 4072, Australia
{shenht,zaiben,zxf}@itee.uq.edu.au

Abstract. Mobile P2P networks have potential applications in many fields, making them a focus of current research. However, mobile P2P networks are subject to the limitations of transmission range, wireless bandwidth, and highly dynamic network topology, giving rise to many new challenges for efficient search. In this paper, we propose a hybrid search approach, which is automatic and economical in mobile P2P networks. The region covered by a mobile P2P network is partitioned into subregions, each of which can be identified by a unique ID and known to all peers. All the subregions then construct a mobile Kademlia (MKad) network. The proposed hybrid retrieval approach aims to utilize flooding-based and DHT-based schemes in MKad for indexing and searching according to designed utility functions. Our experiments show that the proposed approach is more accurate and efficient than existing methods.

1 Introduction

Mobile communication technology continues to proliferate in recent years, and mobile P2P network has been suggested as a potential solution for efficient data sharing, message transferring, information retrieval etc. As a consequence, many new applications naturally arise in telecommunication, commercial and civilian environments, such as mobile phone file sharing, commercial advertisements (ads) broadcasting [1] and traffic estimation [2], making them a focus of current research. Mobile devices (peers) in mobile P2P networks interact during physical encounters in the real world and engage in short distance wireless exchanges of data. Typically, mobile P2P networks enable direct real-time sharing of services and information among distributed peers. In contrast to wired P2P networks that are composed of static peers, mobile P2P networks are subject to the wireless bandwidth, limitations of transmission range, and highly dynamic network topology, giving rise to new challenges for research on routing, search, data

* This research has been supported by the National Grand Fundamental Research 973 program of China under Grant No.2004CB318204, Australian research grant (ARC DP0773483) and National Natural Science foundation of China under Grant No.60873051.

consistency maintenance, etc. The following two scenarios motivate our research on search in mobile P2P networks.

Scenario 1: Retrieving commercial ads information. A mobile P2P network can be constructed for dispatching instant and latest commercial ads. For example, staffs in a supermarket or petrol station can use their handheld devices, e.g., PDA and handphone, to publish the *instant* ads on today's promotion and discount information. Customers of interest with mobile devices can share the information with others when they are moving on roads because of the incentive mechanism. Mobile peers in an initiative mode can search other mobile peers' information under this environment to retrieve commercial ads of their interests. In such a mobile network, the influence of instant ads can be broadcasted and increased to more customers and wider areas.

Scenario 2: Retrieving traffic information. Many centralized traffic information systems have been deployed to provide traffic information of a region for mobile users, e.g. drivers, by broadcasting the information into air. In such traditional centralized mobile networks, each mobile peer has to communicate with the central server. However, many drivers have experienced the common woe of slow response or losing signals on their mobile devices when the bandwidths are most needed. Consequently, traffic information may not be updated timely and continuously, often leading to unpleasant traffic jams. In mobile P2P networks, peers can generate realtime text messages regarding the traffic and share them with other peers to understand the traffic situations just in time. If a mobile peer is heading to city, he/she can initiatively send a query to the mobile P2P network constructed by all the active mobile devices in the area to get the current traffic information of the city. In such an environment, the query is first forwarded to the mobile peers having city traffic information and then the results are routed back the querying peer. Having timely traffic information, mobile peers can avoid traffic jams and improve the traffic situations of the city.

In this paper, we study the problem of search in mobile P2P networks, where each published message is a document. We use the boolean model (i.e. exact match) to retrieve information in mobile P2P networks. In traditional structured P2P networks, Distributed Hash Table (DHT) [3] based data retrieval techniques, given a key, the query will be routed to a specific peer, which is responsible for storing the value associated with the key. However, due to the peers' frequent mobility, the DHT-based retrieval technique alone is not expected to perform equally well in mobile P2P networks due to high DHT maintenance cost. In unstructured P2P networks, the flooding-based technique is the most popular data retrieval one. Flooding entails message processing at every peer. However, it is expensive in terms of communication cost and computational resources in mobile P2P networks. To achieve efficient retrieval, we propose a Hybrid Retrieval approach (HR), which is automatic and economical. In the mobile network, a region partition service can be deployed to be known to all participating peers since vehicle-based peers have the GPS function. The region covered by the network is divided into subregions that construct a mobile Kademlia (MKad) network. HR is designed to utilize flooding-based and DHT-based schemes for quick search.

The main contributions of this paper include: First, we present a novel mobile DHT based on subregions: MKad and its maintenance mechanisms. Second, we propose an effective and efficient search approach named Hybrid Retrieval (HR) in mobile P2P networks. HR is self-adaptive to choose either flooding-based scheme or DHT-based scheme for efficient indexing and searching by analyzing their expected search cost according to cost functions in mobile P2P networks. Third, we confirm the effectiveness and efficiency of our methods by conducting an extensive performance measured by search recall, query latency, query routing efficiency.

The remainder of this paper is organized as follows: Section 2 reviews the related work. In Section 3, we propose a mobile DHT based on Kademlia. We present a hybrid retrieval approach in Section 4. In Section 5, we show the experimental results. Finally, Section 6 concludes the paper with a discussion about our future work.

2 Related Work

There are few existing works on search in mobile P2P networks. The Geographic Hash Table (GHT) system [4] has been initially developed for data storage in sensor networks. Its benefits would be that a responsible cell may be empty without any negative influences, where a perimeter used for storing the location information would simply be outside of the responsible cell. Thus, the robustness is increased. In contrast, GHT may increase complexity of the scheme in combination with a higher network load for maintaining the information in the system. GHT stores information in those peers that form a perimeter around a point. This point in Hierarchical Location Service (HLS) [5] would be the center of the responsible cell. The peer closest to this point is responsible for regularly sending the information to the other members of the perimeter. If the closest peer fails, another member of the perimeter will sent an update after a timer has expired and a new closest peer can be elected. The geographic location system (GLS) [6] is a scalable location service that performs the mapping of a node identifier to its location. GLS can be combined with geographic forwarding to implement unicast.

Rybicki et al. [2] proposed a new paradigm to implement traffic information systems, using an infrastructure-based P2P network consisting of vehicles. This approach has several advantages over traditional VANET-based systems. In addition, it might also be possible for both VANET and P2P based systems to coexist and complement each other. Wolfson et al. [7] proposed a novel search algorithm called Rank-Based Broadcast (RBB) based on device proximity for discovery of local resources in mobile P2P networks. Eriksson et al. [8] presented Cabernet for delivering data to moving vehicles, which uses WiFi access points encountered during drives for network connectivity and provides a beneficial way to use the WiFi networks from moving vehicles.

3 A Novel Mobile DHT

3.1 Preliminaries

Most file queries are for highly replicated objects in P2P networks [9]. However, queries for rare objects are also substantial [10]. Both studies correctly reflect different aspects of the Zipfian distributions. The head of the Zipfian popularity distribution was shown in [9], and hence the query requests are measured based on the objects that match the top 50 queries seen. On the contrary, [10] focuses on the long tail of the distribution. Individual rare objects in the tail may not be requested frequently. However, these queries represent a substantial fraction of the workload, and are worth optimizing. The popularity distribution of a file-sharing workload is flatter than what we would expect from a Zipfian distribution [11]. Therefore, both frequent and rare objects are important for retrieval.

3.2 Region Partition

For an administrative region covered by a mobile P2P network, we divide it into a number of subregions based on system requirements in a recursive manner for maintenance. The region is first divided into two half subregions (R_e and R_w) based on the north/south direction (i.e. longitude), where R_e and R_w are respectively represented by 1 and 0; and then for R_e and R_w, they are also divided into two half subregions (R_n and R_s) based on the east/west direction (i.e. latitude), where R_n and R_s are respectively represented by 1 and 0 as well. The above procedure that is shown in Fig. 1 is recursively processed until the differences in longitude and latitude of a subregion are both less than given thresholds LO and LA. Consequently, the whole administrative region is divided into multiple geographical subregions, which form the network topology. Each subregion is represented by a unique region ID, and each peer uses this embedded service to keep the location information of all subregions in the whole network. Since the network topology consists of geographical subregions, each subregion is responsible for a set of keys of data objects. The keys of data objects are distributed among the subregions, such that each *key* is mapped to a subregion R. This key-to-subregion mapping is known to all peers by using a variety of Kademlia [3] named Mobile Kademlia (MKad). Note that the key-to-subregion mapping is used in MKad, while the key-to-peer mapping is utilized in the original Kademlia.

In MKad, each subregion has a unique ID and each key is a 160-bit identifier. Every document is associated with a home subregion, where the document is initially stored. To assign $\langle key, value \rangle$ pairs to particular subregions, MKad relies on a notion of distance between a subregion ID and a key identifier, where *key* is a 160-bit identifier of a word or document and *value* is a word or document itself. Given a subregion ID R and a *key*, MKad defines the distance between them as their bitwise exclusive or (XOR) interpreted as an integer, $d(R, key) = R \bigoplus key$. Since XOR is unidirectional, it ensures that all lookups for the same key converge along the same path, regardless of the originating peer. Thus, caching $\langle key, value \rangle$ pairs along the lookup path alleviates the load of hot spots. In MKad, there are four subregions with respect to a object: query subregion,

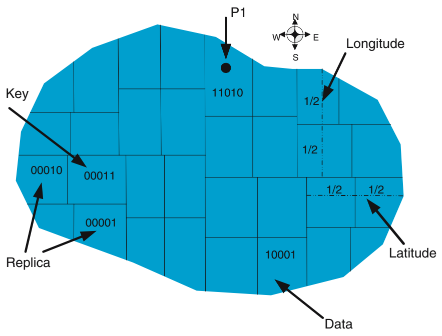


Fig. 1. The procedure of region partition ($k = 2$, k is the number of closest subregions to the key)

Algorithm 1. Maintaining the *MKad* network

```
Input: $\mathcal{R}$ is a set of regions
Output: the MKad network
// Publish rare words
1 foreach subregion $R \in \mathcal{R}$ do
2     Gather new words $\mathcal{W}$ of $R$ based on gossip
3     foreach word $w \in \mathcal{W}$ do
4         $R$ publishes the word $w$

// Drop indexes of popular words
5 foreach subregion $R \in \mathcal{R}$ do
6     $R$ count $\#requested$ of each word
7     foreach word $w \in \mathcal{W}$ do
8         if $w's\#requested \geq \Theta_{th}$ then
9             Drop the popular word $w$'s index
```

key subregion, replica subregion and home subregion in our proposed approach. For example, they are respectively regions: 11010, 00011, {00010, 00001} and 10001 as shown in Fig. 1.

3.3 Aggregation for a Word

To obtain the aggregation for a word w in a subregion, there are two possible approaches. One is gossip-based approach, where peers exchange their current aggregates with their neighbors till the aggregates almost converge to the global values; the other is hierarchy-based approach, where peers come into being a hierarchy and pass the aggregates in a bottom-up manner along the hierarchical path. Hierarchy-based approach requires a super peer mechanism, which is hard to be implemented in mobile P2P networks. Therefore, we utilize gossip-based approach to obtain w's aggregation.

When a peer is moving out of a subregion, its aggregation results and cached documents are transferred to some peers within the subregion which have the following features: 1) low mobility speed; 2) close to the center of the subregion, or moving towards the center; 3) enough cache space. A peer with low mobility and locating near the subregion's center is probably selected to store the aggregates, since it will leave the subregion in the near future with a low probability. In our proposed scheme, gossip-based aggregation has two functions: identifying rare words and publishing new words into MKad. We also differentiate mobile peers by their movements, which could be *intra-subregion* movement or *inter-subregion* movement. For an intra-subregion movement, a peer moves only within the same subregion, thus the overheads are trivial. For an inter-subregion movement, on the contrary, a peer moves out of its initial subregion to a neighboring subregion. Peers need to check their positions periodically to detect an inter-subregion movement.

3.4 MKad Network Maintenance

For each peer in mobile P2P networks, its storage space is limited and thus it is not able to store excessive objects. On the other hand, it is also expensive to maintain too many popular objects in the MKad network due to the limited bandwidth. Popular objects are expected to be dropped from the MKad network if they can be easily and quickly found by flooding-based techniques. Here we use gossip-based aggregation to decide if a word is popular or rare. We present an algorithm shown in Algorithm 1 for maintaining the MKad network. For new words, they are often rare. Thus they are published in the MKad network (lines 2-4). Given a word, if its number of request is not less than Θ_{th} where Θ_{th} is a threshold, the word is regarded as a popular one. Then, the indexes of popular words are dropped from the MKad network (lines 5-9).

4 Hybrid Retrieval

For information retrieval in mobile P2P networks, generally two search schemes can be applied: flooding-based scheme and DHT-based scheme. We first present their cost models, followed by the hybrid retrieval algorithm which can automatically select the more efficient scheme for a query word.

For the flooding-based scheme, the total expected search cost for a query word w is given in [12]: $M^{\mathbf{u}}(w) = \sum_{i=1}^{N} C_{u_i} Pr(X > u_{i-1})$, where a search strategy with TTL values $\mathbf{u} = [u_1, u_2, \cdots, u_N]$, X is the object w's location, C_{u_i} is the search cost of with TTL value u_i, and $u_0 = 0$ is assumed. C_{u_i} was mentioned in [13] that the number of messages incurred with a TTL value of u_i is roughly $u_i + \beta {u_i}^2$, where β is a constant with respect to the network parameters. The search policy that minimizes this cost reads as following:

$$F(w) = M^{\mathbf{u}^*}(w) = \arg\min_{\mathbf{u}\in U} \sum_{i=1}^{N} C_{u_i} Pr(X > u_{i-1}) \tag{1}$$

where U denotes the set of all admissible search strategies (TTL sequences), i.e., all vectors $\mathbf{u}$ such that $u_i < u_{i+1}$ for all $1 \leq i \leq N-1$. Formula 1 can be solved backward in time using standard dynamic programming techniques [12].

For DHT-based scheme, the search cost can be approximately modeled as below:

$$D(w) = \frac{\|R_w - R_o\|}{\gamma} + \sum_{R_i \in \mathbb{R}} \frac{\|R_i - R_o\|}{\gamma} y(R_i, w) \tag{2}$$

where R_o is the originating subregion for the query w, R_w is the subregion that is responsible for the index of w according to MKad, $y(R_i, w)$ is the boolean function returns 1 if R_i publishes w, or 0 otherwise, γ is the transmission range, and $\|.\|$ represents the distance between two subregions.

Based on a simple flooding algorithm presented in Algorithm 2, Algorithm 3 outlines our method - HybridRetrieval. When a peer sends a query q which may contain multiple words, the peer generates a query message containing the following two fields: 1) its identity, and 2) the set of query words. If the search cost of flooding-based scheme is not greater than that of DHT-based scheme (line 5), it floods the query with a given TTL L to determine if any of its neighbors have query answers (line 5). If not, the requesting peer can use the MKad protocol

Algorithm 2. Flooding

```
   Input: $P_o$ is the query originator, $R$ is a
          subregion, $q$ is a query, $L$ is a TTL
   Output: search results $\mathcal{S}$
   // $R_o$ is the originating subregion
1  if $R$ is not $R_o$ then
2      foreach peer $P \in R$ do
3          Retrieve results $\mathcal{S}$ for $q$ from $P$
4  else
5      $\mathbb{P} = \{P_o\}$
6      while $L - - > 0$ do
7          $\mathbb{P}$=Get peers from $\mathbb{P}$ with
           flooding-based scheme
8          foreach peer $P \in \mathbb{P}$ do goto 3
9  Return gossip-based aggregation results to $P_o$
```

Algorithm 3. HybridRetrieval

```
    Input: $P_o$ is the query originator, $q$ is a query
    Output: search results $\mathcal{S}$
1   foreach word $w \in q$ do
2       if $w$ is indexed in MKad then
3           $q$'s key subregion $R_{key} = R_w$
4           break
    // $w$ is the $1^{st}$ word of $q$ indexed in MKad
5   if $F(w) \leq D(w)$ then Flooding($P_o$, $R_o$, $q$, $L$)
6   else
7       Achieving the home regions $\mathcal{R}_{home}$ via $R_{key}$
8       foreach $R_h \in \mathcal{R}_{home}$ do
9           Routing the query $q$ to $R_h$
10          Flooding($P_o$, $R_h$, $q$, 1)
```

to get a subregion (R_{key}) of query word according to the query (line 7). The home subregion of w is determined by searching the subregion whose region ID is the closest to the key of w. Peers that locate outside the home subregion drop the query message without further processing. Flooding is restricted within a subregion for quick query response and savings in network bandwidth (line 10).

A peer that routes the query message towards the destination subregion checks its location in the message's header to determine if it is within that subregion. The first peer that receives the query message inside the destination subregion floods the message within the subregion to locate the peers holding the documents including the requested key words. Each peer in that subregion processes the query message to determine if it has the requested documents. When the documents are located, the response is sent back to the original requesting peer and the query process expires.

5 Experimental Studies

5.1 Experimental Setup

For the simulations, a discrete event simulator NS-2 (*http://www.isi.edu/nsnam /ns/*) is employed with the IEEE 802.11 MAC layer. The NS-2 simulation model simulates peers moving in an unobstructed plane. GPSR [14] is used as the wireless routing protocol. We have modified it to provide routing to subregions instead of specific destinations by forwarding the packet towards the subregion and using the flooding-based scheme inside the subregion.

To measure the performance of our search schemes, we simulate the algorithms on various mobile network topologies by varying the number of peers from 100 to 1000, mean speeds from 1 to 20 m/s, and motion regions from 1500m × 1500m to 4000m × 4000m. The default parameters: # of peers is 400, # of document objects is 10,000, transmission range of a peer is 250m, motion region is 2000m × 2000m, mean speed is 10m/s, length or width of a region is 250m.

5.2 Data Set

We utilize TREC data from the 2GB Web track (WT2G) to simulate documents (ads or traffic information), and their associated words as their descriptive tags.

The top n words within each document are associated with the tags that are valid in describing the document. The words used to describe a document are drawn from its corresponding Web document. It makes the simulations more realistic. We use word frequency to simulate the strength of description that a word has in a document. The relative popularities of documents are arbitrarily assigned according to a Zipf distribution.

5.3 Retrieval Models

A series of experiments are conducted to study several combinations of routing query messages and search in mobile P2P networks. In our experiments, we used simulations to evaluate HR and compared its performance to three other search models in vehicle-based mobile P2P networks. The same partition service of a region is utilized in the four models, which is known to all participating peers.

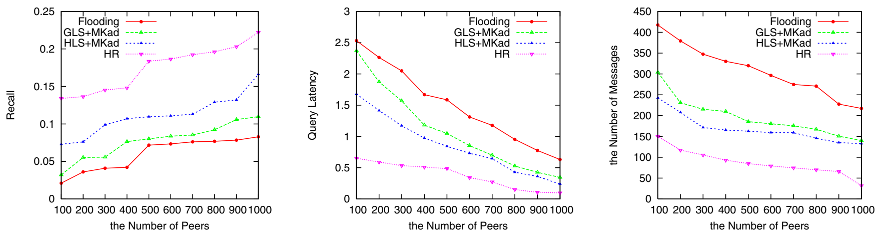
The four search models read as follows: 1) **Flooding**: Search using the flooding-based scheme alone. 2) **GLS+MKad**: Grid Location Service (GLS) [6] builds an ad hoc network's area by using a fixed grid. Each peer maintains 1- and 2-hop neighbor lists. The neighbor lists are built from piggyback messages and indicate a peer's location as well as other parameters like the peer's 1-hop neighbors. In addition, an appropriate cell size is carefully selected regarding the transmission range of the peers. Each peer in a cell knows about all other peers in the same cell. 3) **HLS+MKad**: Hierarchical Location Service (HLS) [5] divides the area of a mobile network into smaller areas called cells and assign each peer a set $\mathbb{S}$ of these cells. Position updates and requests are sent to (possibly different) subsets of $\mathbb{S}$. The selection of the subset depends on a hierarchical grouping of the cells of $\mathbb{S}$ and the position of the peer which computes the subset. The intersection of two subsets computed for the same peer is non-empty in HLS. 4) **HR**: All the subregions construct the MKad network. HR chooses either flooding-based scheme or DHT-based scheme for efficient indexing and searching by analyzing their expected search cost according to cost functions in mobile P2P networks.

GLS and HLS are both position-based routing approaches and do not support search in mobile P2P networks. Thus, we integrate the MKad function into them and use it to get the relevant peers for a given query. Furthermore, GLS and HLS are utilized to route between a response peer and a requesting peer.

5.4 Evaluation Methodology

We adopt three performance metrics to evaluate retrieval in mobile P2P networks, i.e., retrieval accuracy, query latency and query routing efficiency.

Recall For automatically-generated queries, it is expensive for us to acquire relevance judgments in mobile P2P networks. Instead, we used the retrieval results from a single large collection as the baseline, whereafter measuring how well the P2P network could reproduce this baseline. The single large collection is the subset of the experimental test data set used to define peer contents, and agreement is measured over all the data retrieved for each query. Although this methodology is not perfect, it is reasonable because distributed retrieval systems are not better than the "single collection" baseline. Accuracy is measured with



(a) Recall (b) Latency (c) Routing Efficiency

Fig. 2. Effect of Peer Densities

forms of set-based *recall* ($R = \frac{|A \cap B|}{|B|}$), where A is the set of the retrieval data in P2P networks, B is the set of the retrieval data using the subset collection of the single test data set. $R(q)$ captures the fraction of relevant data a retrieval algorithm is able to identify and present to the user.

Query Latency. The query latency is the elapsed time for a query to get its response. It records the overall query processing time.

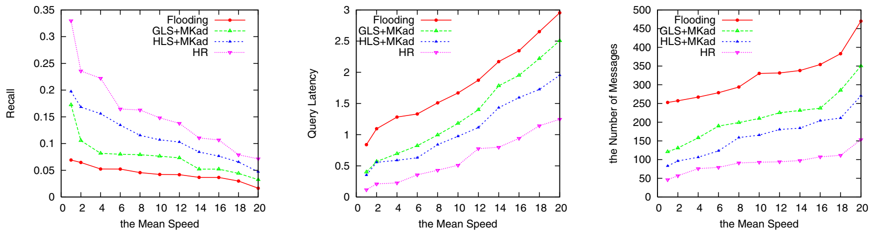
Query Routing Efficiency. Query routing efficiency is usually measured by the average number of query messages routed for each query in mobile P2P networks.

5.5 Experimental Results and Analysis

Experiment 1: Effect of Peer Densities We consider the effects of peer densities on query recall, latency and routing efficiency of different methods. Fig. 2(a), 2(b) and 2(c) show that HR yields the best performance on query recall, latency and routing efficiency. HR uses the proposed utility functions and carefully selects query techniques: flooding and MKad, which can significantly improve the three metrics when the query is initialized and forwarded between peers. HR uses the flooding-based technique to search popular words and uses GPSR to return retrieval results. On the other hand, HR uses MKad to locate rare words and uses the flooding-based scheme to find them only in single subregions.

However, GLS and HLS only use MKad to locate words. In the static P2P networks, the query recall of DHT techniques is relatively high. However, the performance drops greatly because mobile P2P networks are highly dynamic. On the other hand, GLS and HLS achieve a much worse query performance than HR because they produce much more update packets leading to higher network load and network congestion in the end. Moreover, there is a tradeoff between the maintenance cost for the MKad network and the query effectiveness regarding GLS and HLS.

Obviously, the flooding-based technique is the worst on three metrics among the four approaches because of three general reasons: 1) it is ineffective and inefficient to return the query answers; 2) it is less effective for locating rare items: 28.1% of all queries receive 5 or fewer results, and 12.3% of queries receive no results in this experiment, although it is highly effective for locating popular words, which are retrieved in large quantities. On the other hand, although it is



(a) Recall (b) Latency (c) Routing Efficiency

Fig. 3. Effect of Mobility Speeds

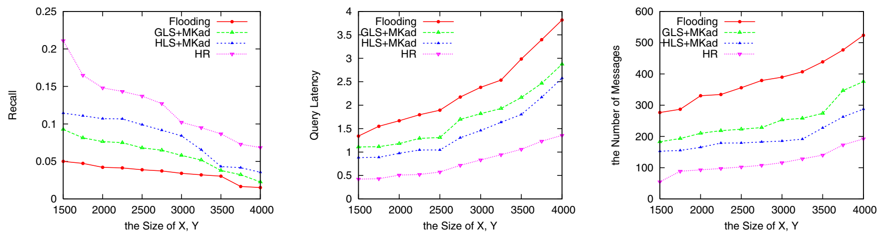
highly efficient for locating popular words, where the queries have good lookup time, it has bad returning time; 3) it is less efficient for locating rare objects and the results have poor response time. For queries that return a single result, the first result arrives after 7.2 seconds on average. For queries that return 5 or fewer results, 5.6 seconds elapsed on average before receiving the first result in this experiment. However, HR uses the designed utility functions to select the query techniques for improving the query recall, latency and routing efficiency. As the peer density increases, HR can construct stabler MKad network to search rare words and locate much more popular words with smaller TTL.

Experiment 2: Effect of Mobility Speeds To evaluate the scalability of HR with respect to mobility speeds, we increase the mobility speeds from 1m/s to 20m/s. We still evaluate three metrics: query recall, latency and routing efficiency of four approaches. The results presented in Fig. 3(a), 3(b) and 3(c) indicate that HR still achieves significantly better query performance than the other three methods.

It is more difficult for the flooding-based mechanism to locate rare words and return retrieval results at higher speed. The network load in GLS and HLS increases with growing peer speed because more updates are sent that lead to worse query routing efficiency. On the other hand, a higher load produces more collisions resulting in re-queries and in unwanted delays in GLS and HLS. Facing to high peer speed, HR still has better query performance than the other three methods because it is based on subregions besides the proposed cost functions. On the other hand, HR construct the MKad network only for rare objects, which is effective and efficient at high speed. On the contrary, both GLS and HLS are not effective and efficient to construct the MKad network for all the objects at high speed.

Experiment 3: Effect of Motion Regions with a Fixed Peer Density We consider the effect of motion regions with a fixed peer density (100 peers per square kilometer) on query performance of four methods. Fig. 4(a), 4(b) and 4(c) show the results of query recall, latency and routing efficiency. Similarly, HR can retrieve significantly better query performance than previously proposed methods. Clearly, the flooding method is the worst among the four approaches, which does not need to be elaborated any further.

To improve the query performance, the proposed utility functions in HR help to carefully select query techniques: flooding and MKad. Moreover, HR uses



(a) Recall (b) Latency (c) Routing Efficiency

Fig. 4. Effect of Motion Regions with a Fixed Peer Density

GPSR to locate response peers and return query results to a requesting peer. Since GPSR makes greedy forwarding decisions using the only information about a router's immediate neighbors, both query latency and query efficiency are improved further. However, GLS and HLS only use MKad to locate words. Besides previous explanations in experiment 1 and 2, their query messages in a larger region must be forwarded to requesting peers and response ones via a longer distance, which is a more difficult task in mobile P2P networks with fast movers.

6 Conclusions and Future Work

In this paper, we gave two scenarios to motivate our research on search in mobile P2P networks. And then we proposed hybrid retrieval policies to utilize flooding-based and MKad-based schemes for quick search in mobile P2P networks. Our experiments show our approach yields better performance. In particular, the query response time and the number of messages per query are reduced substantially without losing accuracy. We believe that possible directions to future work include some research topics such as extending our method with cooperative cache in mobile P2P networks, evaluating the MKad network's performance and maintenance cost, improving the simple flooding algorithm, and examining the impact of diverse duplication mechanisms in our retrieval model.

References

1. Chen, Z., Shen, H.T., Xu, Q., Zhou, X.: Instant advertising in mobile peer-to-peer networks. In: ICDE (to appear, 2009)
2. Rybicki, J., Scheuermann, B., Kieß, W., Lochert, C., Fallahi, P., Mauve, M.: Challenge: peers on wheels - a road to new traffic information systems. In: MOBICOM, pp. 215–221 (2007)
3. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
4. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., Shenker, S.: Ght: a geographic hash table for data-centric storage. In: WSNA, pp. 78–87 (2002)
5. Kiess, W., Fussler, H., Widmer, J., Mauve, M.: Hierarchical location service for mobile ad-hoc networks. SIGMOBILE Mob. Comput. Commun. Rev. 8(4), 47–58 (2004)

6. Li, J., Jannotti, J., Couto, D.S.J.D., Karger, D.R., Morris, R.: A scalable location service for geographic ad hoc routing. In: MOBICOM, pp. 120–130 (2000)
7. Wolfson, O., Xu, B., Yin, H., Cao, H.: Search-and-discover in mobile p2p network databases. In: ICDCS, p. 65 (2006)
8. Eriksson, J., Balakrishnan, H., Madden, S.: Cabernet: vehicular content delivery using wifi. In: MOBICOM, pp. 199–210 (2008)
9. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable. In: SIGCOMM, pp. 407–418 (2003)
10. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The case for a hybrid p2p search infrastructure. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279, pp. 141–150. Springer, Heidelberg (2005)
11. Gummadi, P.K., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: SOSP, pp. 314–329 (2003)
12. Chang, N.B., Liu, M.: Revisiting the ttl-based controlled flooding search: optimality and randomization. In: MOBICOM, pp. 85–99 (2004)
13. Baryshnikov, Y., Coffman, E., Jelenkovic, P., Momcilovic, P., Rubenstein, D.: Flood search under the california split rule. Operations Research Letters 32(3), 199–206 (2004)
14. Karp, B., Kung, H.T.: Gpsr: greedy perimeter stateless routing for wireless networks. In: MOBICOM, pp. 243–254 (2000)

An Algorithm for Unrestored Flow Optimization in Survivable Networks Based on *p*-Cycles

Adam Smutnicki

Chair of Systems and Computer Networks,
Wrocław University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wrocław, Poland
adam.smutnicki@pwr.wroc.pl

Abstract. This paper provides compound algorithm for Unrestorable Flow Optimisation (UFO) problem formulated for computer networks protected by p-cycles, created on the base of mathematical model and solution approaches proposed in our complementary paper [1]. Components of the algorithm have been selected carefully and then experimentally tested in order to compose the best final algorithm. Results of the wide computer tests on common benchmarks have been also presented as well as some practical conclusions following from the research made.

Keywords: Computer network, survivability, optimization, p-cycles, UFO problem.

1 Introduction

In [1] we have shown that Unrestorable Flow Optimization (UFO) problem, as a complicated task, may be decomposed into several auxiliary subproblems with dedicated solution methods. The mentioned above paper outlines also solution methods without practical realizing comments. This paper refers chiefly to solution algorithms derived from theory, [1] and their numerical properties tested for the best practical performance. Paper [1] is complementary to this research.

2 Algorithm Components

Consider the problem of allocating demands into p-cycles while restoring the net flow in case of failure. The problem can be modeled by Multiple Knapsack Problem (MKP) which is strongly $\mathcal{NP}$-hard. One can find several approximate and exact algorithms for MKP in [2]. In our research we use a greedy algorithm for MKP, based on principle of best fit rule – element is packed into this knapsack in which most of free space is left. This algorithm has pseudo-polynomial complexity $O(nm)$ (n is a number of elements and m is a number of knapsacks) acceptable in practice, taking into account that overall solution is approximate. Up to now, research have been made using commonly Greedy algorithm for MKP; quite recently there have been proposed and analyzed profits of usage branch-and-bound algorithm for MKP in UFO problem in [3].

Next subproblem, described in [1], consists in determining the maximum spare capacity of all p-cycles. For non-disjoint p-cycles there can be written set of linear equations or inequalities, where criteria function, as mentioned in [1] is to maximize sum of residual capacity of all p-cycles. This forms a set of linear equations and can be solved using algorithm dedicated Linear Programming task. Simplex method, as most popular and easiest to use, has been chosen.

At the end, for each demand there have to be prepared a set of alternative routing paths. Because demands are defined from source to target nodes, k-SPD algorithm based on Dijkstra's Shortest Path Algorithm, can be used [4].

2.1 Generation of p-Cycles

The optimal configuration of p-cycles protecting the network can be either constructed from scratch or selected among certain subset of p-cycles called cycle-candidates set. The number of all possible cycles in network with the full mesh topology (i.e. full undirected graph) is $\omega(2^V)$, i.e. is rising faster than 2^V, where V is the number of vertices in the net, see [5] for detail. Thus, assuming usage of the step-by-step search technique, even for quite small networks, the total number of cycle candidates is usually too big to store all of them in the computer memory and then to check any of them. Therefore, two alternative approaches one can propose next: (a) use pre-generated set of cycle candidates, of limited reasonable cardinality, generated in advance, in off-line mode, (b) generate on-line cycle candidates on demand, whereas the number of generated cycles is a priori unknown. The former approach is called *pre-generation technique*, whereas the latter – *enumeration technique*, [5,6]. In both cases a "good" p-cycles are expected from the generation algorithm. The problem of generation cycle components is nontrivial and has been studied among others in [5,6,7,8,9,10].

The presented paper provides research made for conception (a). Application of (b) remains an open research task. A decision process, which particular p-cycles are used for protection, will be done by tabu search (TS) algorithm, see Section 2.2 for detail. In order to evaluate the impact of generation scheme onto overall solution quality, we tested four different generation algorithms, to compare results obtainable for different sets of p-cycles. A lot of algorithms generate larger set of p-cycles by extending a small one with the help of some transformations. We refer here to the well-known algorithm Straddling Link, proposed first time in [9]. It is quite simple and never generates more cycles than number of spans in network. In [7,11] there have been described several different algorithms providing rules for transformations from simple p-cycles to complex ones. Among them, three have been chosen as the most promising: SP-Add (Span Add), Expand and Grow, described in [7].

2.2 Tabu Search

It has been shown, [1], that UFO problem is strongly $\mathcal{NP}$-hard. Thus the use of heuristic or metaheuristic algorithms is fully justified. We decided to apply Tabu Search (TS) technology, because of good results achieved for many other applications in optimization. Referring the reader to foundations of TS, [12], we present here only implementations of its crucial elements in our case. Each *solution* of our problem will be represented by current configuration of p-cycles (a subset of components selected from the whole candidates set) plus current set of routing paths (only one path is selected for each flow demand). *Quality of the solution* is measured by the amount of unrestored flow, expressed as fraction of total flow. The whole evaluation of UFO value consists of calculation of capacities for chosen p-cycles (an LP task) and determining which demands can be restored (by solving the sequence of MKP instances), see [1] for detail.

Because our solution consists of two inhomogeneous structures (p-cycles and routing paths) the *move* (transformation) from the current solution can be either *p-cycle move* or *routing path move.* All moves, generated from the current solution, generate its *neighborhood.* Let us describe these transformations more precisely. First, because of small change philosophy, we assume that transformed solutions differ from original one by single element. For transformation performed on routing configuration, we had to choose solution which will not increase drastically the size of TS neighborhood. For example, if there are three alternative routing paths for some demand, two new neighboring solutions will be generated for this demand — each of them corresponds to one of two remaining routing path (except this already used). Thus, having k alternative paths and d demands, $(k-1)d$ new neighboring solutions will be generated. For p-cycles transformations we have defined three possible changes (three neighbors) for each component of p-cycle : add one p-cycle to current configuration from candidates set (not used yet), remove one p-cycle from current configuration, exchange one p-cycle from current configuration into one from candidates set (not used yet). Each new element in neighborhood is generated by performing only one of mentioned transformations. Whole neighborhood coming from p-cycles set transformation is done by performing for each p-cycle in current configuration following steps: (a) remove a component p-cycle; (b) add n times randomly chosen with weights (value of AE ([11]) metric as weight) p-cycle from candidates set; (c) exchange current cycle component into one from candidates set being not far than $|l|$ steps from actual. p-Cycles in candidates set are sorted decreasingly by value of AE metric. We see that parameters l and n values has significant influence on size and type of generated TS neighborhood. Next, we will call l a size of p-cycles neighborhood and n as number of randomly chosen p-cycles.

The starting solution of TS is created by using the following principles: (1) for each demand, the first (shortest) path on the candidate list is set, (2) current configuration of p-cycles contains single component, the first one from the list of candidates (with greatest value of metrics AE).

Tabu list (short term memory of the search history) is considered as the crucial control element of TS, since it is responsible for proper behavior of the whole algorithm (it prevents cycling). In our implementation we assume that tabu list stores the move made, both types of moves on the common list. The move to be performed is considered as tabu if its inverse move is stored on tabu list. The form of inverse move is easy to define for the moves introduced earlier.

Selection of TS *control parameters* is discussed in Section 3.4.

3 Computational Results

The quality of proposed (metaheuristic) algorithm will be evaluete in computer test on common benchmarks.

3.1 Common Benchmarks

For network optimization tasks there exists a library, called SNDlib [13], providing a set of network topologies, as well as all necessary information about flows and possible routes. SNDlib consists of several network topologies, which are in most cases real networks, or at least advanced projects, e.g.: COST-266, germany50, poland, france. Test described in this paper have been performed using COST-266 topology, because it is quite similar to COST-239 (COST-239 is the project older than COST-266). COST-239 was used in most of papers in literature dealing with p-cycles optimization.

3.2 Generation Technology Test

First performed test compares the behavior of algorithm depending on p-cycles generation algorithm. Four, mentioned in Section 2.1, algorithms have been tested. Results are presented on the Fig. 1, where one routing path for each demand is used. First and most obvious conclusion is that the Grow algorithm finds final result using smaller number of TS iterations. Grow also descends toward final result quicker that other algorithms. We see that best results are achieved using p-cycles set from Grow algorithm, but it is quite interesting that for this combination, the TS algorithm needed less time to find better result than using other p-cycles algorithm. So the configuration of parameters giving the best

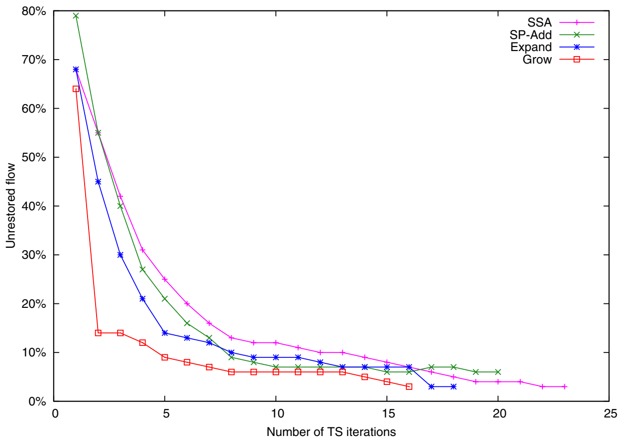


Fig. 1. Comparison of p-cycles generation algorithms for one routing path for each demand

Table 1. Comparison of parameters of p-cycles generated by four mentioned algorithms, for COST-266 topology

	SLA	SP-Add	Expand	Grow
number of p-cycles	45	96	89	1214
av. value of AE metrics	1.27	1.49	1.56	1.74
av. p-cycle length	7.20	11.04	13.01	21.38
av. number of straddling-spans	1.17	2.88	3.96	8.48
average generation time in seconds	0.0019	0.0076	0.0061	2.65

results produce them in shorter time than other, worse configurations. Only one disadvantage of this solution is that the time needed to generate p-cycles candidates set using Grow algorithm is significantly longer than for other algorithms. Average time needed for generation sets of p-cycles and parameters of those sets for test COST-266 topology, for tested four algorithms arepresented in Tab. 1.

3.3 Alternative Routing Paths Test

In Section 3.2 there have been analyzed influence of chosen p-cycles generation algorithm on final result. But as mentioned in [1], this is not only one control parameter. Second one, no less important is the number of alternative routing paths for each demand. Moreover during whole optimization in TS, the most useful path is chosen for each demand, so giving more optional paths, algorithm receives wider range of possible solutions. On Fig. 2 there have been presented comparison of p-cycles generation algorithm but for three alternative routing

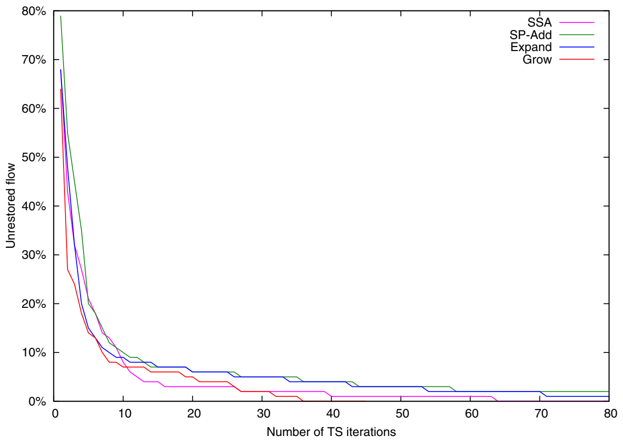


Fig. 2. Comparison of p-cycles generation algorithms for three routing path for each demand

Table 2. Comparison of example results depending on used p-cycles generation algorithm and number of alternative routing paths; presented values stands for % of unrestorable flow

	SLA	SP-Add	Expand	Grow
1 kSPD	3.84	6.70	6.87	3.68
3 kSPD	0.67	1.25	1.28	0.35

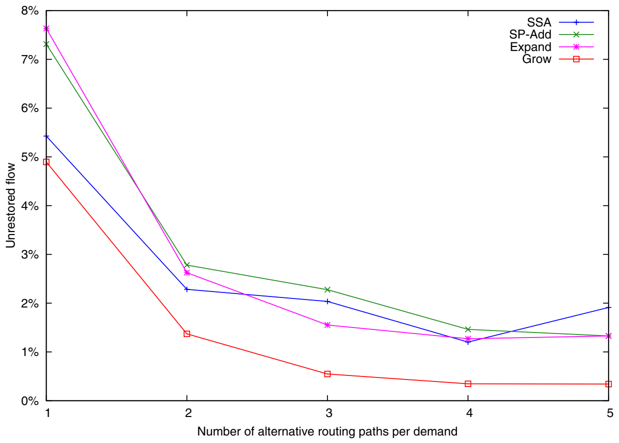


Fig. 3. Comparison of results for each p-cycles generation algorithm depending on number of alternative routing paths for each demand

paths for each demand. Comparing Fig. 1 and Fig. 2 one can notice, that giving more alternative paths improve the way that overall algorithm works. First improvement is that for each p-cycles generation algorithm TS descending toward final result is quicker. Second conclusion is that, TS is working longer (bigger number of TS iterations till stop condition). It is because using more alternative paths, allow TS algorithm to tune more the final result. Also, as previously, the best p-cycles generation algorithm is Grow.

Presented results on Fig. 1 show only general behavior of two described options. Exact results are presented in Tab. 2. Analyzing those results one can see that using three alternative paths improve the quality of final result several times. So without any additional resources, only changing the routing, adjusting it for chosen set of p-cycles, we can improve the level of final protection.

Comparing results of mentioned test with tests for only one routing path, we see that only changing flow in network, we can highly improve the protection performed by p-cycles set. So it is highly recommended that p-cycles optimization should be accompanied with dedicated routing.

One can ask, why not use more routing paths, if it gives such improvement. The answer is presented on Fig. 3. We can see that the best improvement of

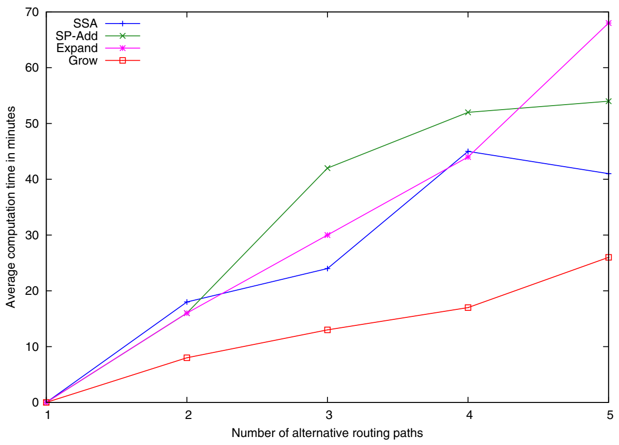


Fig. 4. Comparison of time needed to find best solution depending on used p-cycles generation algorithm and number of alternative routing paths

quality of final result is when changing from one to three alternative routing paths for each demand. Adding more paths till five, practically do not change the quality of received final result. What is important, adding more alternative paths increase the time of computation – Fig. 4.

For more that three routing paths, possible improvement is not worth because of amount of time needed to finish computations. What more, the increase of time needed for all algorithms except Grow is higher than for Grow.

Analyzing presented results for time of computation and time needed to generate a set of p-cycles (presented in Section 3.2) we can see, that using Grow is highly recommended because of best preformed results and shortest computation time, even taking into account time needed to generate p-cycles set. This additional time is several hundred times smaller that time saved in TS.

3.4 Finding the Optimal Configuration

In whole process of finding final solution several elements are important. Two main were analyzed and discussed in Section 3.2 and 3.3. Other are: size of tabu list, number of TS iterations till algorithm stops, size of p-cycles neighbourhood, number of random p-cycles configuration change. All mentioned parameters have been tested within defined ranges, to find which of them and how much influence the quality of final result.

During tests we have notices that TS algorithm was falling into cycle nearby found local minimum. In this case, we decided to try to block this cycles, as this prevents algorithm for further search. Those cycles came from situation, when reaching local minimum, TS while generating neighborhood received most of solutions with the same value of criteria function. Also there have to be mentioned, that in basic structure tabu list does not protect before cycles. We decided to filter TS neighborhood – solutions having the same value of criteria function were not analyzed. Results presented in this paper compare version of TS with and without filtering.

Because of limited size of this paper, we are unable to present all results and comparisons, so we decided to present only conclusions drawn from performed tests. The conclusions are:

1. we have not notices influence of size of p-cycles neighborhood on quality of final result;
2. there exists dependency of quality of final result on number of randomly chosen p-cycles configuration – till the value of 5 it increases the quality, bigger values practically do not change anything;
3. for algorithm without filtering the neighborhood one can notice link between final solution quality and length of tabu list – longer list improves the quality, but influence is quite small;
4. for algorithm with filtering the neighborhood we have not noticed link between the quality of final solution and tabu list size – probably tabu list blocks only one step back and filtering process prevents algorithm from backtracking;
5. we have notices direct influence of number of TS iterations till stop on quality of final result – there is an improvement till value of 20, after this value there is no increase in quality;
6. analyzing all parameters, we have notices that algorithm with filtering neighborhood, generally gave better results that without filtering;
7. in all cases for option with filtering solutions and without, best results were received using Grow p-cycles generation algorithm;
8. quite surprising was that second best results were received using SSA p-cycles generation algorithm – the one that generates simplest and most basic p-cycles;
9. computations using Grow algorithm gave better results, because p-cycles set generated by this algorithm consists of p-cycles form SSA algorithm (which was second best) and some additional p-cycles with different feature, which improved the quality of final solution, over single set from SSA.

3.5 Final Solution Quality Test

In Section 3 there have been presented received results. The best parameters configuration have been identified and discussed. In this section we will analyze the quality of received results. Because for now one have performed such research like described in this paper, there is no point which we can refer to. Test have been performed for TS algorithm with filtering and without. Both of them have started from same basic solutions, which "unrestorability" was 64.15%. In Tab. 3 there have been presented average received results for 100 repeats. Also there are some statistic parameters of results sample. Time of computations for TS with filtering was ca. 30% longer that without. We can see that both versions have reached 0.0% of unrestorability and both have done it four times. Median

Table 3. Comparison of results quality for TS algorithm with filtering and without, for optimal algorithm configuration, for COST-266 topology; values are value of unrestorability in %

	without filtering	with filtering
av. result value	0.62	0.43
best value	0.00	0.00
best value found no. of times	4	4
worst value	5.11	3.82
median	0.35	0.35
standard deviation	0.87	0.42
av. deviation	0.49	0.17

is the same, version with filtering have much lower deviation from average value. Additionally there have been done T-Student statistic tests to confirm that both samples differ each other. Tests were positive.

4 Conclusions

We have presented a set of algorithms building overall solution for UFO problem formulated in our paper [1]. Presented algorithms are first ever developed for UFO problem. Performed tests analyze wide range of parameters influencing the quality of final result. Notice, we have proposed a joint optimization of routing and selecting p-cycles, which most of authors in literature have avoided, called them too complex. We have found configuration of the net ensuring unrestorable flow below 1% of total flow in network, and in many situations achieving 100% of restorability. Taking into account that this optimization is done in advance, we have enough time to find best solution. We have also confirmed that the combination of p-cycles with different size (generated by Grow algorithm) provides best restorability.

References

1. Smutnicki, A., Smutnicki, C.: Flow Optimization in Survivable Networks Based on p-Cycles. Submitted for International Conference on Computational Science (ICCS 2009) (2009)
2. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)
3. Smutnicki, A.: Branch-and-bound Algorithm for Multiple Knapsack Problem in the Unrestorable Flow Optimisation Problem. In: Proceedings of 23rd IAR Workshop on Advanced Control and Diagnosis, Coventry, UK (November 2008)
4. Pióro, M., Medhi, D.: Routing, Flow, and Capacity Design in Communication and Computer Networks. Elsevier Inc., San Francisco (2004)

5. Schupke, D.: An ILP for Optimal p-Cycle Selection Without Cycle Enumeration. In: Proceedings of Eighth IFIP Working Conference on Optical Network Design and Modelling (ONDM 2004), Ghent, Belgium (February 2004)
6. Wu, B., Yeung, K., Xu, S.: ILP Formulation for p-Cycle Construction Based on Flow Conservation. In: Proceedings of Global Telecommunications Conference, 2007. GLOBECOM 2007, Washington, DC, USA, pp. 2310–2314 (November 2007)
7. Doucette, J., He, D., Grover, W.D., Yang, O.: Algorithmic Approaches for Efficient Enumeration of Candidate p-Cycles and Capacitated p-Cycle Network Design. In: Proceedings of Fourth International Workshop on Design of Reliable Communication Networks, pp. 212–220 (October 2003)
8. Chang, L., Lu, R.: Finding Good Candidate Cycles for Efficient p-Cycle Network Design. In: Proceedings of 13th International Conference on Computer Communications and Networks, pp. 321–326 (October 2004)
9. Zhang, H., Yang, O.: Finding Protection Cycles in DWDM Networks. In: Proceedings of IEEE International Conference on Communications, pp. 2756–2760 (2002)
10. Lo, K., Habibi, D., Rassan, A., Phung, Q.V., Nguyen, H.N., Kang, B.: A Hybrid p-Cycle Search Algorithm for Protection in WDM Mesh Networks. In: Proceedings of ICON 2006. 14th IEEE International Conference on Networks, pp. 1–6 (September 2006)
11. Grover, W.D.: Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET, and ATM Networking. Prentice Hall PTR, New Jersey (2003)
12. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Pusblishers, Dordrecht (1997)
13. Orlowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0 — Survivable Network Design Library. In: Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium (April 2007), http://sndlib.zib.de

Unrestored Flow Optimization in Survivable Networks Based on *p*-Cycles

Adam Smutnicki

Chair of Systems and Computer Networks,
Wrocław University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wrocław, Poland
adam.smutnicki@pwr.wroc.pl

Abstract. This paper deals with Unrestorable Flow Optimisation (UFO) problem in networks protected by p-cycles. This novel protection technique is used as the efficient tool for ensuring survivability of computer networks. In this paper there have been formulated mathematical model of UFO problem, discussed its theoretical properties, and proposed the original solution algorithm based chiefly on metaheuristics. The algorithm combines k-shortest paths method, multi knapsack problem, p-cycles generator, linear programming and some local search procedures.

Keywords: computer network, survivability, optimisation, p-cycles, UFO problem.

1 Introduction

Survivability of computer networks and systems is located among the most important subjects in modern computer engineering and science. This research topic embraces the wide spectrum of particular technological and theoretical problems derived from computer architecture area, network topology, communication protocols, transmission, coding, cryptography, etc. The topology of the computer network has crucial meaning for its survivability, since physical creation of the net links is much more time-consuming and troublesome than producing a new (or spare) device, furthermore faults of network links and nodes are still the common problem. The idea of usage *p*-cycles is quite new, but have been widely developed among recent years. *p*-Cycles are very favorable in comparison with traditional ring or mesh topologies. In this paper, we present and discussed the new optimization problem, generated by the concept of network protection by using *p*-cycles, where the criteria is not a cost, but the level of restorability ensured by using currently available net resources.

This paper is organized as follows. Section 2 provides brief introduction into *p*-cycles idea. Section 3 describes, in detail, new problem of unrestorable flow optimization, called hereinafter the UFO problem. Section 4 discusses some its properties and solution methods. Conclusions one can find in Section 5. Because of the strong NP-hardness of UFO problem, this paper introduces basically the mathematical model, then discusses its essential properties including layer decomposition and computational complexity and, at the end, provides a wide spectrum of solution methods. Particular algorithms, as well as experimental results based on standard benchmarks are shown in separate our paper [1].

2 Background of *p*-Cycles

Traditionally, the most popular solution for providing restorability in computer networks was to use either *ring* or *mesh* topology. Ring offers short restoration time as well as simple restoration scheme, however its design and operation is rather complex and the usage of total transport bandwidth is inefficient. Mesh is easy to design, optimize and operate, but have greater than ring restoration time. Mesh networks don't require as much spare capacity as rings, because in the restoration process capacity demand can be split between different links. On the other hand, rings are so efficient in the restoration process because there are no need to search for restoration path. Obviously, there is a great need to find topology, which aggregates all advantageous properties of mesh and ring networks. This idea was fully realized in the concept of p-cycles, that means "fast as ring", "efficient as mesh", preconfigurable and protected.

In normal non-failure state, routing for flow demands between pairs of nodes, is done using one of many routing techniques. Set of p-cycles is formed in advance while configuring network, to be ready to use in case of any failure and perform real-time recovery. p-Cycles are not an ordinary cycles. Let us consider a mesh network and choose some cycle (Fig. 1 *a*). In classical cycle protection approach, this cycle protects all spans being "on-cycle". In the paper [2] it is shown that cycle established on mesh network protects also "straddling spans", i.e. spans between cycle nodes, but not belonging to the cycle (Fig. 1 *b*). Observe, that in case of failure of "straddling span" the arc of cycle can be used to transfer whole flow from this failed span. This property allows one to extend protection provided by p-cycles on straddling spans as well. Fig. 1 *c* shows which spans cannot be protected using this properties.

In case of failure of "on cycle" span, there is one path which can be used to transfer flow (Fig. 2 *a*). But for failure of "straddling span" there are two different paths, which can be used for recovery process. Arc of the cycle can be used as a path, or both arcs to achieve lower load on links (Fig. 2 *b* and *c*). Because without using any additional links and spare capacity we achieve much higher level of protection, protected are not only cycle spans but also "straddling spans". "Straddling spans" have twice the leverage of an on-cycle span in terms of efficiency because when they fail, the cycle itself remains intact and can thereby offer two protection paths for each of unit of protection capacity. This spans are not limited to be inside a cycle, each span between two nodes of the cycle is protected by this idea.

Notice, we do not need any additional spare capacity to protect "straddling spans", because the spare capacity from ring spans is used to protect those spans. This means that we can protect much more spans and link capacity using the same amount of spare capacity as in the ring model. Thus, under the some costs, we can achieve higher level of network survivability.

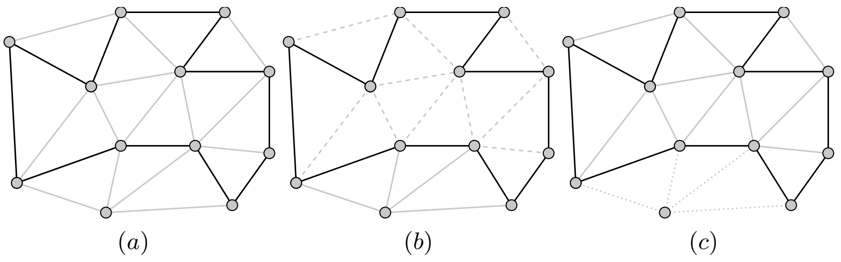


Fig. 1. *p*-Cycles in the mesh network

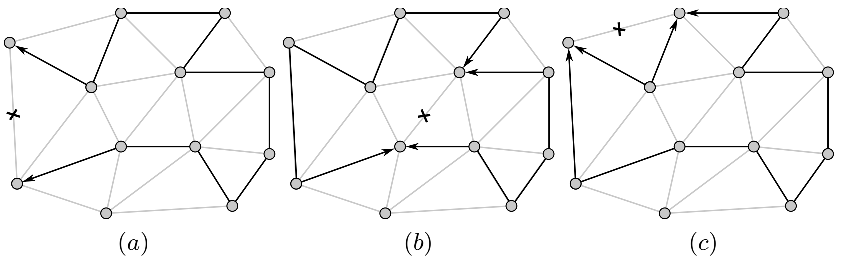


Fig. 2. *p*-Cycles protection schemes for various types of failure

3 The UFO Problem

In the literature, most papers dealing with survivable networks concentrate on ensuring 100% survivability. But only few authors have considered (not only mentioned) problem, where 100% of restorability may not be achievable. For networks protected by *p*-cycles, in [3] there has been proposed problem in which *the level of restorability* is maximized, assuming fixed amount of spare capacity.

Another paper [4] mentions about different idea — minimization of unrestorable flow in the network with fixed capacities, where no additional capacity is necessary, since restoration is done within available spare capacity, left after optimization of all flow demands. This problem, called by us Unrestorable Flow Optimisation (UFO), will be studied extensively in our paper. Below, we define it formally as follows. (Notation based on [5] and [6] will be used.)

We have given: network topology, link capacities, traffic demand matrix, candidate paths for demands, *p*-cycles configuration. Optimization over working flows in normal non-failure state of the network is done, for protection in the case of single link failure. The objective is to minimize the unrestored flow, i.e. flow that due to limited link capacity cannot be restored using *p*-cycles.

Indices

$e, l = 1, \ldots, E$ network links (spans)
$d = 1, \ldots, D$ demands

$p = 1, \ldots, P_d$	candidate paths for flows realizing demand d
$q = 1, \ldots, Q$	p-cycles
$s = 1, \ldots, S$	failure states

Constants

$\mathcal{D}$	set of all demands
$\mathcal{E}$	set of all spans
$\mathcal{Q}$	set of all p-cycles
$\mathcal{Q}_e \in \mathcal{Q}$	set of p-cycles which can be used for restoration span e
$\mathcal{D}_e \in \mathcal{D}$	set of demands using span e
δ_{edp}	$= 1$, if link e belongs to path p realizing demand d; 0 otherwise
h_d	— volume of demand d
c_e	— capacity of link e
β_{eq}	$= 1$, if link e belongs to p-cycle q; 0 otherwise
ϵ_{eq}	$= 1$, if p-cycle q can be used for restoration of link e; 0 otherwise i.e. link e either belongs to p-cycle q or is a straddling span of q
γ_{eq}	— coefficient of restoration paths provided for failed link e by an instance of p-cycle q ($= 1$ for an on-cycle link; $= 0.5$ for a straddling span; $= 0$ otherwise)

Variables

x_{dp}	$= 1$ if demand d uses path p; 0 otherwise (binary)
f_e	— load of link e associated with working demands
y_{deq}	$= 1$ if demand d uses path p-cycle q for restoration in the case of failure of link e; 0 otherwise
z_{de}	$= 1$ if demand d is not restored in the case of failure of link e; 0 otherwise
g_{el}	— load of link e associated with p-cycle in the case of failure of link l

Overall optimization criteria function is:

$$\min_{y,z} U(y, z; x, \mathcal{Q}) = \min_{y,z} \sum_e \sum_d z_{de} h_d, \tag{1}$$

with constraints:

$$z_{de} + \sum_q \epsilon_{eq} y_{deq} = A_{de},\ e = 1, \ldots, E,\ d = 1, \ldots, D \tag{2}$$

$$g_{el} = \sum_d \sum_q B_{eldq} y_{dlq},\ e = 1, \ldots, E,\ l = 1, \ldots, E \tag{3}$$

$$g_{el} \le s_e, \quad e = 1, \ldots, E, \quad l = 1, \ldots, E, \tag{4}$$

where A_{de}, B_{eldq}, are constants used to make equations more clear for fixed x:

$$f_e = \sum_d \sum_p \delta_{edp} x_{dp} h_d, \quad e = 1, \ldots, E, \tag{5}$$

$$A_{de} = \sum_p \delta_{edp} x_{dp}, \quad e = 1, \ldots, E, \quad d = 1, \ldots, D, \tag{6}$$

$$B_{eldq} = \sum_p \delta_{ldp} x_{dp} \beta_{eq} \gamma_{lq} h_d, \quad e = 1, \ldots, E,$$
$$l = 1, \ldots, E, \quad d = 1, \ldots, D, \quad q = 1, \ldots, Q, \tag{7}$$

and:

$$s_e = c_e - f_e, \quad e = 1, \ldots, E. \tag{8}$$

Auxiliary variable s_e, defined in (8) and used in constraint (4), will be called *residual spare capacity*, left after fulfilling all traffic demands from set $\mathcal{D}$ transfered over paths determined by x. Auxiliary constant A_{de} takes only binary values, $A_{de} \in \{0, 1\}$, which means that either there is no possibility to restore demand d in case of failure of span e (then $z_{de} = 1$ and all $y_{deq} = 0$, $q = 1, \ldots, Q$) or there exists possibility of restoring d in case of failure of span e using one chosen p-cycle q (then $y_{deq} = 1$ and $z_{de} = 0$). The formulated optimization problem is a Mixed Integer Linear Programming (MILP) task. However, because of the problem size, and well-known weakness of the general MILP solution methods, we give up MILP techniques and transform the problem into new one.

Using equation (2) in (1) and substituting equation (3) into inequality (4), the problem receives the following form:

$$\min_{y,z} U(y, z; x, \mathcal{Q}) = \min_y \sum_e \sum_d (A_{de} - \sum_q \epsilon_{eq} y_{deq}) h_d$$
$$= \sum_e \sum_d A_{de} h_d - \max_y \sum_e \sum_d \sum_q \epsilon_{eq} h_d y_{deq}, \tag{9}$$
$$\sum_d \sum_q B_{ledq} y_{deq} \le s_l, \; e = 1, \ldots, E, \; l = 1, \ldots, E. \tag{10}$$

In these equations index e stands for span being damaged; when another index l is used also for spans, it refers to other span which has been influenced by damaged span e.

First element in equation (9) is constant and does not have any influence on the form of optimal solution, but only on its value. Problem (9) described by second element with constraint (10), can be solved by a sequence of problems known in literature as Knapsack Problem (KP) or Multiple Knapsack Problem (MKP). Constraint (10) determines whether there is a need to consider KP either or MKP case. Transformation of (9) – (10) into MKP is shown below.

Consider right side of equation (9) for fixed span e in a form:

$$\max_y \sum_d \sum_q C_{edq} y_{deq}, \tag{11}$$

where:

$$C_{edq} = \epsilon_{eq} h_d, \tag{12}$$

with constraints:

$$\sum_d \sum_q B_{ledq} y_{deq} \le s_l, \quad l = 1, \ldots, E. \tag{13}$$

For fixed e, exists $d \times q$ binary decision variables y_{deq}. So the criteria function (11) corresponds to optimal packing Q knapsacks with elements $1, 2, \ldots, D$. In order to match constraint (13) with constraints from MKP [7], the new notion r_q will be used, called hereafter *residual capacity of p-cycle* q. Value r_q stands for maximum flow which can be added to all spans in p-cycle q, without exceeding residual capacity of the spans. Calculation of p-cycle residual capacities is a complex problem – we will discussed it, in detail, in Section 4.3. For further considerations, we assume that capacity of p-cycle meet constraint $r_q > 0$ and:

$$s_l = \sum_q \beta_{lq} r_q. \tag{14}$$

Substituting (14) in constraint (13) one can obtain:

$$\sum_d \sum_q B_{ledq} y_{deq} \le \sum_q \beta_{lq} r_q, \quad l = 1, \ldots, E. \tag{15}$$

Condition (15) is then transformed into sequence of $Q \times E$ conditions in the following form:

$$\sum_d B_{ledq} y_{deq} \le \beta_{lq} r_q, \; q = 1, \ldots, Q, \; l = 1, \ldots, E, \tag{16}$$

what makes stronger constraint that original (15). Summing inequalities (16) by sides for each l (this is relaxation) inequality (13) will be received. Observe in definition (7) that value B_{ledq} does not depend on l, so:

$$B_{ledq} = B^*_{edq} = \sum_p \delta_{edp} x_{dp} \gamma_{eq} h_d, \tag{17}$$

because (16) is fulfilled obviously for $\beta_{lq} = 0$, while this condition have to be fulfilled additionally for $\beta_{lq} = 1$. Using (17), constraint (15) can be transformed into:

$$\sum_d B^*_{edq} y_{deq} \le r_q, \quad q = 1, \ldots, Q. \tag{18}$$

for those l for which $\beta_{lq} = 1$. It corresponds to missing constraint for knapsacks capacities $1, \ldots, Q$ in MKP (constraints (18) are identical for all l). If $Q = 1$ (special case) MKP is simplified and takes form of KP.

Finally, value:

$$\max_y \sum_e \sum_d \sum_q C_{edq} y_{deq}, \tag{19}$$

can be calculated step by step, analyzing possibility of restoring flow f_e using p-cycles after the failure of span e, for all $e \in \mathcal{E}$ such $f_e > 0$. If only $f_e = 0$ nothing have to be restored — $y_{deq} = 1$, $z_{de} = 0$, $e \in \mathcal{E}$ because $h_d = 0$ for $d \in \mathcal{D}$. So (19) can be finally written in a form:

$$\max_y \sum_{e \in \mathcal{E}_+} \sum_{d \in \mathcal{D}_e} \sum_{q \in \mathcal{Q}_e} C_{edq} y_{deq}, \tag{20}$$

where: $\mathcal{E}_+ = \{e \in \mathcal{E} : \; f_e > 0\}$.

4 Solution Proposal

Solution proposal tends to decomposition of the problem (1) – (8) leading to the series of known, simpler cases.

4.1 Decomposition

In the context of Section 3 one can propose quite natural layer decomposition of optimization problem into several sub-problems. At the begin we have the network with known topology, span capacities and costs and a set of flow demands. Each demand determines flow transfer between pair of nodes (from source to destination). In order to satisfy these demands routing paths have to be found. This problem is known in the literature as multicomodity flow (MCF). MCF problem requires the set of paths (for example k shortest paths k-SP) between specified pair of nodes, among which MCF selects the set of the best ones ([5]). Observe, the configuration of paths satisfying demands need be advantageous for p-cycles configuration. Our overall aim is to find the optimal set of p-cycles; this can be done by metaheuristic algorithm, which goes through the solution space by certain search trajectory verifying candidates on p-cycles. Successive p-cycles in this space are pre-generated by using a reasonable generator. One among many described in literature can be used.

For fixed set of p-cycles and fixed routing paths realizing demands we have to solve the UFO problem. Its optimal solution is created by independent checking of restorability for each span with nonzero flow. If this span transfers single commodity flow, its restorability case can be simply evaluated. Otherwise, if the span transfers multicommodity flow, the Multiple Knapsack Problem (MKP) is used to find optimal restoration scheme (described briefly in Section 4.2). Both cases require the evaluation of so called *spare capacity of the cycle*. For disjoin cycles, inside current p-cycles configuration, such evaluation can be made independently and the problem is not troublesome. If cycles inside current p-cycles configuration are not disjoint, the problem of finding real *spare capacity of cycles* can be written as special linear programming (LP) problem — described in Section 4.3.

4.2 Minimisation of Unrestorable Flow

Let us consider criteria function (1) in the mathematical model from Section 3. In order to calculate its value we need to known matrix z_{de}, where h_d is known in advance as the input data for this problem. According to the description in Section 3, all routing paths have to be chosen for this calculations (realizing all flow demands) and dedicated configuration of p-cycles also have to be known, with its residual capacities calculated. Algorithm for calculation of z_{de} values is presented in Lis. 1.1. All symbols are consistent with those from Section 3. This algorithm refers to equations (9) – (10). It examines all spans $e \in \mathcal{E}_+$ in the network, whether whole flow from D_e can be restored using p-cycles $\mathcal{Q}_e$ in case of failure of each particular span.

Listing 1.1. Pseudocode of algorithm for calculation z_{de} values.

```
for $e \in E$ do
  begin
    $(R_e, U_e) = checkRestoriationPosibility(e, D_e, Q_e)$;
    for $d \in R_e$ do $z_{de} = 0$;
    for $d \in U_e$ do $z_{de} = 1$;
  end;
```

The key role plays function *checkRestoriationPosibility*(); it determines whether all flow demands can be restored or not in case of failure of span e. If all flow demands cannot be restored, the function returns two sets of: R_e for restorable, and U_e for unrestorable demands, minimising the total amount of unrestored flow.

One can distinguish four following scenarios in case of failure of span e (We say that particular span belongs to p-cycle, if it is either on-cycle span or a straddling-span on this p-cycle.):

1. span e does not belong to any p-cycle, so $\mathcal{Q}_e = \emptyset$; thus definitely flow from span e cannot be restored, so $y_{deq} = 0$, $q = 1, \ldots, Q$ and in consequence $z_{de} = 1$, $d \in D_e$;
2. flow on span e consists of only one commodity $|\mathcal{D}_e| = 1$ and span e belongs to one p-cycle, so $|\mathcal{Q}_e| = 1$; flow can be either restored or not, depending on residual capacity of this p-cycle;
3. flow on span e is a multicommodity flow, $|\mathcal{D}_e| > 1$ and span e belongs to one p-cycle, so $|\mathcal{Q}_e| = 1$ — flow can be restored or not depending on residual capacity of this p-cycle;
4. flow on span e is a multicommodity flow and span e belongs to more than one p-cycle, so flow can be restored or not depending on residual capacity of those p-cycles.

Case (1) is obvious. Case (2) is simple – flow can be restored if $h_d \leq r_q$ (h_d is a value of flow for demand d, r_q is a residual capacity of p-cycle q). In case (3), the total flow for span e can be restored if $\sum_{d \in D_e} h_d > r_q$. In this case, the selection of demands to be restored is significant — it is the optimization problem, modeled by using Knapsack Problem. The most complex case (4) is modeled by MKP, as already mentioned.

4.3 *p*-Cycles Residual Capacities

According to description in Section 3 there is no possibility to add any spare capacity, only existing spare capacity in working spans can be used. This assumptions generates several additional constraints. The most important is a decision problem how much of spare capacity have to be assigned to each of used p-cycles, in situation when at least two cycles have common span — Fig. 3. In this situation sum of p-cycles capacities cannot exceed span spare capacity. Additionally maximization of sum of whole p-cycles spare capacity is desired. Mathematical formulation of this problem is described below. Let us define:

r_q — capacity of p-cycle q;
s_e — available spare capacity on span e;
r_q^{max} — maximum potential capacity of p-cycle q (defined in equation (21));
β_{eq} $= 1$ if span e belongs to p-cycle q;

Maximum potential capacity of each p-cycle is bounded by value:

$$r_q^{max} = \min\{s_e : e = 1, \ldots, E, \quad e \in q\}. \tag{21}$$

For each span we have constraint:

$$\sum_q r_q \beta_{eq} \le s_e, \quad e = 1, \ldots, E \tag{22}$$

Total amount of p-cycles capacity should be maximised:

$$\max \sum_q r_q, \quad q = 1, \ldots, Q \tag{23}$$

taking into account constraints:

$$0 \le r_q \le r_q^{max}, \quad q = 1, \ldots, Q \tag{24}$$

The problem (23) – (24) is a typical linear programming task, so simplex method can be recommended here to solve it.

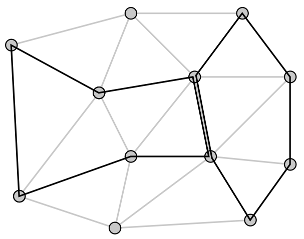


Fig. 3. Example of configuration where two p-cycles have common span

4.4 Problem Complexity

Two described previously problems, namely the finding of z_{de} values and calculation of residual capacities, are only activities performed during the evaluation of the current (fixed) p-cycles and the current routing paths configuration. In the pessimistic case, we have to solve certain MKP problem(s), which is, according to [7], strongly $\mathcal{NP}$-hard. So the process of evaluation single solution is strongly $\mathcal{NP}$-hard, too. Note, that the generation of the set of p-cycles candidates (in the local search algorithm) appears to be also quite complicated problem, see [1] for detail, because the number of possible cycles in network has non-polynomial character; the similar remarks refers also to generation of the set of paths – candidates for realizing routes for each demand.

5 Conclusions

We have formulated problem of increasing computer network survivability by using p-cycles as an combinatorial optimization problem. The problem alone has been neither discussed nor solved in the literature, yet. Although the problem can be modeled as a general MILP task, presented by us transformations show that it is quite complicated case. We have also shown that problem is strongly $\mathcal{NP}$-hard, which suggests that the most suitable solution methods should rely on metaheuristic approaches. We have also shown original decomposition method and then proposed highly dedicated algorithm for each particular subproblem. Due to limited size of the paper, we have presented here the primal part of extensive studies made for the problem stated – detailed analyzes of the algorithm as well as wide experimental research on common benchmark test are presented in the complementary paper [1], also submitted for this conference.

References

1. Smutnicki, A., Smutnicki, C.: An Algorithm for Flow Optimization in Survivable Networks Based on p-Cycles. Submitted for International Conference on Computational Science, ICCS 2009 (2009)
2. Grover, W.D., Stamatelakis, D.: Cycle-oriented Distributed Preconfiguration: Ring-like Speed with Mesh-like Capacity for Self-planning Network Restoration. In: Proceedings of ICC 1998 IEEE International Conference on Communications, Atlanta, Georgia, USA, June 1998, pp. 537–543 (1998)
3. Grover, W.D.: Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET, and ATM Networking. Prentice Hall PTR, New Jersey (2003)
4. Smutnicki, A., Walkowiak, K.: Modeling Flow Allocation Problem in MPLS Network Protected by p-Cycles. In: Peringer, P., S., J. (eds.) Proceedings of 42nd Spring International Conference on Modelling and Simulation of Systems, Hradec nad Moravici, Czech Republic, MARQ, April 2008, pp. 35–42 (April 2008)
5. Pióro, M., Medhi, D.: Routing, Flow, and Capacity Design in Communication and Computer Networks. Elsevier Inc., San Francisco (2004)
6. Chołda, P., Jajszczyk, A., Wajda, K.: The Evolution of the Recovery Based on p-Cycles. In: Proceedings of the 12th Polish Teletraffic Symposium PSRT 2005, Poznań, Poland, pp. 49–58 (September 2005)
7. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)

Simulation of Complex Systems

Hierarchical Modelling and Model Adaptivity for Gas Flow on Networks

Pia Bales, Oliver Kolb, and Jens Lang

Technische Universität Darmstadt
Schloßgartenstr. 7, 64289 Darmstadt, Germany

Abstract. We are interested in the simulation and optimization of gas transport in networks. Different regions of the network may be modelled by different equations. There are three models based on the Euler equations that describe the gas flow in pipelines qualitatively different: a nonlinear model, a semilinear model and a stationary also called algebraic model. For the whole network, adequate initial and boundary values as well as coupling conditions at the junctions are needed. Using adjoint techniques, one can specify model error estimators for the simplified models. A strategy to adaptively apply the different models in different regions of the network while maintaining the accuracy of the solution is presented.

Keywords: model adaptivity, adjoint equations, gas flow.

1 Introduction

During the last years, there has been intense research in the field of simulation and optimization of gas transport in networks [2,3,4,8,9]. The equations describing the transport of gas in pipelines are based on the Euler equations, a hyperbolic system of nonlinear partial differential equations, mainly consisting of the conservation of mass, momentum and energy. The transient flow of gas may be described appropriately by equations in one space dimension. For the whole network, adequate initial and boundary values as well as coupling conditions at the junctions are needed.

Although solving one-dimensional equations does not pose a challenge, the complexity increases with the size of the network. Thus, we present a hierarchy of models that describe the flow of gas in pipelines qualitatively different: The most detailed model we use consists of the isothermal Euler equations (continuity equation and momentum equation). A common simplification of the momentum equation leads to a semilinear model, which is only valid if the velocity of the gas is much less than the speed of sound, that is, $|v| \ll c$. Further simplifications lead to the steady state model. Obviously, simplified models are sufficient in network regions with low activity in the gas transport, while sophisticated models should be used to resolve high solution dynamics accurately. Since the whole network behaviour can change in both space and time, an automatic steering of the model hierarchy is essential. Existent software packages like SIMONE [11] may use stationary as well as transient models for the simulation. However, for the simulation process one model has to be chosen. The different models are introduced in Sect. 2. The modelling of the network as well as the boundary and coupling conditions are presented in Sect. 3.

In order to estimate the model error of the simplified models, that is, of the semilinear and the steady state model with respect to some quantity of interest, one has to solve adjoint systems on the network. For the adjoint equations, appropriate coupling conditions are required, which are introduced in Sect. 4. There, we also present a strategy, how to decide in which regions of the network which model has to be used to reduce the complexity of the whole problem, whereas the accuracy of the solution is maintained. We give numerical examples of this algorithm in Sect. 5.

2 Model Hierarchy

In this section, we introduce a hierarchy consisting of three different models. Each model results from the previous one by making further simplifying assumptions [1]. The most complex model is the nonlinear model followed by the linear model. The most simple model used is the algebraic model (see Fig. 1).

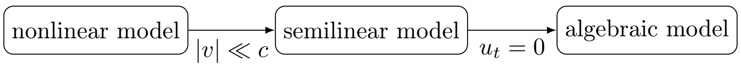


Fig. 1. Model hierarchy

2.1 Nonlinear Model

The isothermal Euler equations, which describe the flow of gas, consist of the continuity and the momentum equation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho v)}{\partial x} = 0, \qquad \frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho v^2)}{\partial x} + \frac{\partial p}{\partial x} = -g\rho h' - \frac{\lambda}{2d}\rho|v|v \tag{1}$$

together with the equation of state for real gases $\rho = \frac{p}{z(p,T)RT}$.

Here, ρ denotes the density, v the velocity of the gas, p the pressure, g the gravity constant, h' the slope of the pipe, λ the friction coefficient, d the diameter of the pipe, R the (special) gas constant, T the temperature of the gas (assumed to be constant) and $z = z(p, T)$ the compressibility factor.

For the sake of simplicity, we assume the pipe to be horizontal and the compressibility factor z to be constant. This results in a simplified equation of state $\rho = \frac{p}{c^2}$ with constant speed of sound $c = \sqrt{RT}$. Since the mass flow M can be traced back to the flow rate under standard conditions ($M = \rho v A = \rho_0 q$), the system can be rewritten in the following way:

$$p_t + \frac{\rho_0 c^2}{A} q_x = 0, \qquad q_t + \frac{A}{\rho_0} p_x + \frac{\rho_0 c^2}{A}\left(\frac{q^2}{p}\right)_x = -\frac{\lambda \rho_0 c^2 |q| q}{2dAp}. \tag{2}$$

Here, ρ_0 and q denote density and flow rate under standard conditions (1 atm air pressure, temperature of $0\,°\mathrm{C}$), A the cross-sectional area of the pipe.

The characteristic speeds are the eigenvalues of the Jacobian of the system, which are $\lambda_{1/2}(u) = v \mp c$. Hence, for subsonic flow, the characteristics travel in opposite directions with characteristic speeds $\lambda_{1/2}$ depending on the velocity of the gas.

2.2 Semilinear Model

If the speed v of the gas is much smaller than the speed of sound, we can neglect the nonlinear term in the spatial derivative of the momentum equation in (1). Together with the equation of state as above, this yields a semilinear model

$$u_t + \mathbf{A}u_x = \psi(u) \tag{3}$$

$$\text{with } u = \begin{pmatrix} p \\ q \end{pmatrix}, \; \mathbf{A} = \begin{pmatrix} 0 & \frac{c^2\rho_0}{A} \\ \frac{A}{\rho_0} & 0 \end{pmatrix} \text{ and } \psi(u) = \begin{pmatrix} 0 \\ -\frac{\lambda\rho_0 c^2|q|q}{2dAp} \end{pmatrix}.$$

For this model the characteristic speeds are $\lambda_{1/2} = \mp c$. Thus, information always travels in both directions with sonic speed.

2.3 Algebraic Model

A further simplification leads to the stationary model: Setting the time derivatives in (3) to zero results in

$$q_x = 0, \qquad \frac{A}{\rho_0}p_x = -\frac{\lambda\rho_0 c^2|q|q}{2dAp}. \tag{4}$$

Thus, q is constant in space and the exact solution for p is

$$p(x) = \sqrt{p(x_0)^2 + \frac{\lambda\rho_0^2 c^2|q|q}{dA^2}(x_0 - x)}\;.$$

Here, $p(x_0)$ denotes the pressure at an arbitrary point $x_0 \in [0, L]$. Setting $x_0 = 0$, that is, $p(x_0) = p(0) = p_{in}$ at the inbound of the pipe, and $x = L$, that is, $p(x) = p(L) = p_{out}$ at the end of the pipe, yields the algebraic model [10].

For the other two models, we computed characteristic speeds at which information propagates in different directions. Since this model is stationary, information given at any place instantaneously influences all other points.

3 Modelling of the Network

We now want to describe the gas flow on networked pipelines. For this purpose, we model the network as a directed graph $\mathcal{G} = (\mathcal{J}, \mathcal{V})$ with edges $\mathcal{J}$ (pipes) and vertices $\mathcal{V}$ (nodes, branching points). Each edge $j \in \mathcal{J}$ is defined as an interval (x_j^a, x_j^b) with a direction from x_j^a to x_j^b. Of course, all intervals are disjoint. Then, for any inner node v, we can define two sets of edges. Let the set of

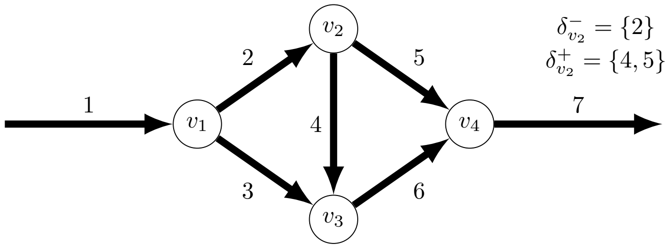


Fig. 2. A small network; the ingoing pipes of node v_2 are $\delta^-_{v_2} = \{2\}$ and the set of outgoing pipes is $\delta^+_{v_2} = \{4,5\}$

ingoing pipes be denoted by δ^-_v, that is, the set of any edge $j \in \mathcal{J}$ with endpoint x^b_j being adjacent to v. Then, analogously, δ^+_v denotes the set of outgoing pipes (see Fig. 2). Inside each pipe, one of the models described above holds. In order to obtain a unique solution, we have to pose coupling conditions at the inner nodes of the network as well as boundary conditions at the sources and sinks.

3.1 Coupling Conditions

A first coupling condition is the conservation of mass at each inner node. Let $v \in \mathcal{V}$ be a node with ingoing pipes $j \in \delta^-_v$ and outgoing pipes $i \in \delta^+_v$. Then, Kirchhoff's law (conservation of mass) yields

$$\sum_{j \in \delta^-_v} q(x^b_j, t) = \sum_{i \in \delta^+_v} q(x^a_i, t)\,.$$

Next, we need further coupling conditions and there are several possibilities. The most common condition used is the equality of pressure at the node v as pointed out in [3]: $p(x^b_i, t) = p(x^a_j, t)\ \forall i \in \delta^+_v, j \in \delta^-_v$.

3.2 Boundary Conditions

Let $\mathcal{J}_{in}$ denote the set of ingoing pipes of the network, i.e. the pipes connecting the sources with the network and let $\mathcal{J}_{out}$ denote the set of outgoing pipelines connected with sinks.

Since for subsonic flow the characteristics of the nonlinear model propagate in different directions and for the semilinear model the characteristics always propagate in reverse directions, one can prescribe the characteristic variables only on opposing sides of a pipe. Thus, there are some limitations on the boundary conditions of the edges.

One possibility is to specify the pressure p at one end of the pipe and the flow rate q at the other. So, we usually prescribe the pressure at $x^a_j, j \in \mathcal{J}_{in}$ (sources) and the flow rate at $x^b_j, j \in \mathcal{J}_{out}$ (sinks).

3.3 Gas Flow on the Network

We can now describe the flow of gas on the network. With the notations $\Omega = \bigcup_{j\in\mathcal{J}}[x_j^a, x_j^b]$ and $Q := \Omega \times (0,T)$, the equations for the nonlinear model read as follows:

$$\begin{array}{ll} u_t + f(u)_x = \psi(u) & \text{in } Q \\ p(x,0) = p_0(x) & \text{in } \Omega \\ q(x,0) = q_0(x) & \text{in } \Omega \\ p(x_i^a,t) = w_i(t) & i \in \mathcal{J}_{in}, t \in (0,T) \\ q(x_i^b,t) = v_i(t) & i \in \mathcal{J}_{out}, t \in (0,T) \\ p(x_i^b,t) = p(x_j^a,t) & \forall v \in \mathcal{V}, i \in \delta_v^-, j \in \delta_v^+, t \in (0,T) \\ \sum\limits_{i\in\delta_v^-} q(x_i^b,t) = \sum\limits_{i\in\delta_v^+} q(x_i^a,t) & \forall v \in \mathcal{V}, t \in (0,T) \\ w_i(t) > 0 & i \in \mathcal{J}_{in}, t \in (0,T)\,. \end{array} \tag{5}$$

For the semilinear and the algebraic model, the equations are analogous to (5) with the corresponding PDE or algebraic equation in the first line. The boundary conditions $p(x_i^a,t), i \in \mathcal{J}_{in}, t \in (0,T)$ and $q(x_i^b,t), i \in \mathcal{J}_{out}, t \in (0,T)$ are determined by control variables/functions $w_i(t)$ and $v_i(t)$. Since the flow rate at the sinks is given by the consumers, the variable that can be controlled by us will only be the pressure at the sources.

4 Adjoint Equations on the Network

A possibility to achieve a compromise between the accuracy of the model and the computational costs is to use the more complex model only when necessary. Using the solution of adjoint equations as done in [5,6], we deduce a model error estimator to measure the influence of the model on a user-defined output functional.

Let the functional M be of the form

$$M(u) = \int_Q N(u)\,\mathrm{d}t\,\mathrm{d}x + \sum_{i\in\mathcal{J}_{in}} \int_0^T N_{x_i^a}(q)\,\mathrm{d}t + \sum_{i\in\mathcal{J}_{out}} \int_0^T N_{x_i^b}(p)\,\mathrm{d}t + \int_\Omega N_T(u)\,\mathrm{d}x. \tag{6}$$

As pointed out in [6], we only need to solve the dual problem of the simplified models in order to obtain a first order error estimator. Let $\xi = (\xi_1, \xi_2)^T$ be the solution of the dual problem of the semilinear model (3) or the algebraic model (4) with respect to the functional M.

For a given solution $u^* = (p^*, q^*)^T$ of the semilinear equations, the adjoint system on the network reads as follows:

$$\begin{array}{ll} \xi_t + \mathbf{A}^T \xi_x = -\partial_u \psi(u^*)^T \xi - \partial_u N(u^*)^T & \text{in } Q \\ \xi(\cdot,T) = \partial_u N_T(u^*(\cdot,T))^T & \text{in } \Omega \\ \xi_1(x_i^a,t) = -\frac{A_i}{\rho_0 c^2} \partial_q N_{x_i^a}(q^*(x_i^a,t)) & i \in \mathcal{J}_{in}, t \in (0,T) \\ \xi_2(x_i^b,t) = \frac{\rho_0}{A_i} \partial_p N_{x_i^b}(p^*(x_i^b,t)) & i \in \mathcal{J}_{out}, t \in (0,T)\,. \end{array} \tag{7}$$

The adjoint system for the algebraic equations is similar to that of the semilinear model, only that the time derivative and the initial conditions vanish. Thus, one cannot measure the influence of the algebraic model at the final time T, which means that the last term of 6, i.e. $\int_\Omega N_T(u)\,\mathrm{d}x$, has to be left out. For the adjoint systems, one also has to specify coupling conditions. Conservation of mass and equality of pressure at the node v yield for the adjoint variables:

$$\begin{aligned} \tfrac{1}{A_i}\xi_1(x_i^b,t) &= \tfrac{1}{A_j}\xi_1(x_j^a,t), && i \in \delta_v^-, j \in \delta_v^+, t \in (0,T)\,, \\ \sum_{i\in\delta_v^-} A_i\xi_2(x_i^b,t) &= \sum_{j\in\delta_v^+} A_j\xi_2(x_j^a,t), && t \in (0,T)\,. \end{aligned}$$

4.1 Error Estimators

We now use the adjoint equations to assess the simplified models with respect to the quantity of interest. Let $u = (p,q)^T$ be the solution of the nonlinear model (2) and $u^h = \left(p^h,q^h\right)^T$ the discretized solution of the semilinear model (3). Then the difference between the output functional of u, $M(u)$, and $M(u^h)$ is

$$\begin{aligned} M(u) - M(u^h) = \int_Q N(u) - N(u^h)\,\mathrm{d}t\mathrm{d}x + \sum_{i\in\mathcal{J}_{in}} \int_0^T N_{x_i^a}(q) - N_{x_i^a}(q^h)\,\mathrm{d}t \\ + \sum_{i\in\mathcal{J}_{out}} \int_0^T N_{x_i^b}(p) - N_{x_i^b}(p^h)\,\mathrm{d}t + \int_\Omega N_T(u) - N_T(u^h)\,\mathrm{d}x. \end{aligned}$$

Taylor expansion of first order yields

$$\begin{aligned} &= \int_Q \partial_u N(u^h)(u-u^h)\,\mathrm{d}t\mathrm{d}x + \sum_{i\in\mathcal{J}_{in}} \int_0^T \partial_q N_{x_i^a}(q^h)(q-q^h)\,\mathrm{d}t \\ &+ \sum_{i\in\mathcal{J}_{out}} \int_0^T \partial_p N_{x_i^b}(p^h)(p-p^h)\,\mathrm{d}t + \int_\Omega \partial_u N_T(u^h)(u-u^h)\,\mathrm{d}x + H.O.T. \end{aligned}$$

with $H.O.T.$ being higher order terms. Inserting the solution ξ of the adjoint system (7), we get a first order error estimator for the model and the discretization error respectively as in [6]:

$$M(u) - M(u^h) \approx \eta_m + \eta_h \tag{8}$$

with the estimators η_m and η_h as follows:

$$\eta_m^{nl-sl} = \int_Q -\xi^T \begin{pmatrix} 0 \\ \frac{\rho_0 c^2 (q^h)^2}{Ap^h} \end{pmatrix}_x \mathrm{d}x\mathrm{d}t \tag{9}$$

$$\eta_h^{nl-sl} = \int_Q \xi^T \left(-u_t^h - \mathbf{A}u_x^h + \psi(u^h)\right)\,\mathrm{d}x\mathrm{d}t. \tag{10}$$

Since the algebraic model can be solved exactly, the discretization error disappears and one only gets an estimator for the model error

$$\eta_m^{sl-alg} = \int_Q -\xi^T \begin{pmatrix} p \\ q \end{pmatrix}_t \mathrm{d}x\mathrm{d}t \tag{11}$$

with ξ being the solution of the adjoint equations either of the semilinear model (7) or of the algebraic model. Here, $u = \left(\begin{smallmatrix} p \\ q \end{smallmatrix}\right)$ denotes the solution of the stationary model (4).

4.2 Adaptive Switching Strategy

With the estimators defined above we may now derive a strategy to switch adaptively between the models. For this, we divide the time interval $(0, T)$ into equal subintervals (T_{k-1}, T_k), $k = 1, \dots, N_B$, with $T_0 = 0$ and $T_{N_B} = T$. Thus, we can split up the computational domain $Q = \Omega \times (0, T)$ into N_B blocks $Q_k = \Omega \times (T_{k-1}, T_k)$, $k = 1, \dots, N_B$, of equal size (see Fig. 3(a)).

We start with simulating the first block Q_1. Each pipe is assigned to one of the three models. Then, we solve the corresponding adjoint system in order to estimate the model error using (9) and (11) respectively. The model error estimator on Q_1 can now be computed for each pipe separately. For the semilinear case (9) this reads as follows.

$$\begin{aligned}
\eta_m &= \sum_{k=1}^{N_B} \int_{Q_k} -\xi^T \begin{pmatrix} 0 \\ \frac{\rho_0 c^2 (q^h)^2}{A p^h} \end{pmatrix}_x \mathrm{d}x\mathrm{d}t \\
&= \sum_{k=1}^{N_B} \sum_{j \in \mathcal{J}} \int_{T_{k-1}}^{T_k} \int_{x_j^a}^{x_j^b} -\xi^T \begin{pmatrix} 0 \\ \frac{\rho_0 c^2 (q^h)^2}{A p^h} \end{pmatrix}_x \mathrm{d}x\mathrm{d}t =: \sum_{k=1}^{N_B} \sum_{j \in \mathcal{J}} \eta_m(k, j)
\end{aligned}$$

with the “local” estimators $\eta_m(k, j)$.

Given a tolerance TOL, one can decide in which pipe the model used is appropriate and in which it is not. We want to accept the model if the relative deviation of the simpler model u^h from the exact solution of the more complex model u is below TOL, that is, $\left|M(u) - M(u^h)\right| / \left|M(u^h)\right| \leq \texttt{TOL}$. Provided that the discretization error is nonsignificant compared to the model error, we can approximate $\left|M(u) - M(u^h)\right|$ by $|\eta_m|$, which yields

$$|\eta_m| \leq \texttt{TOL} \left|M(u^h)\right| . \tag{12}$$

Just like the error estimator η_m, we can evaluate the target functional M at every pipe $j \in \mathcal{J}$ and every time interval (T_{k-1}, T_k), $k = 1, \dots, N_B$ individually, giving $M_{k,j}$, that is, $M(u^h) = \sum_{k=1}^{N_B} \sum_{j \in \mathcal{J}} M_{k,j}(u^h)$. Thus, for inequality (12) to hold, it suffices to claim

$$|\eta_m(k, j)| \leq \texttt{TOL} \left|M_{k,j}(u^h)\right| , \quad \forall k \in \{1, \dots, N_B\}, j \in \mathcal{J} . \tag{13}$$

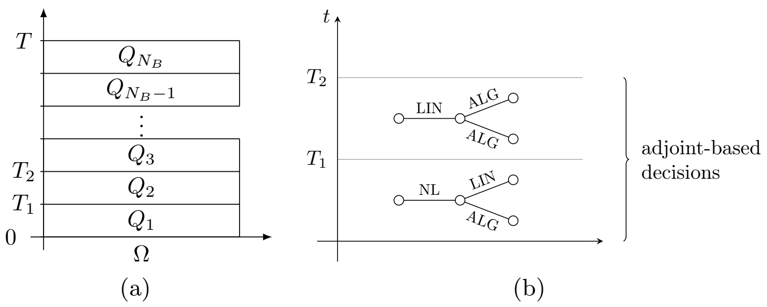


Fig. 3. (a) Partition of the computational domain; (b) Scheme of the adaptive switching (ALG = algebraic model, LIN = semilinear model, NL = nonlinear model)

If any of the estimators $\eta_m(k,j)$ violates (13), the computation of this time interval has to be repeated and the models used in these pipes have to be exchanged by a more complex model. For those pipes of which the estimators fulfil inequality (13), one can evaluate the estimators "downwards". If these also fulfil (13), a more simple model may be used in the next time step. For a scheme of the switching strategy, see Fig. 3(b).

5 Numerical Results

We give an example of the algorithm for a small network. It consists of nine pipes Le1 to Le9, one source Qu, four inner nodes M1 to M4 and three sinks Se1 to Se3 (see Fig. 4(a)).

All pipes have a diameter of 1m and a roughness of $5 \cdot 10^{-5}$ m. The lengths of all except two pipes is 10km. The pipes Le4 and Le6 are both 5km long. The simulation time totals $T = 14400$ s with time step size $\Delta t = 5$ s. The block size was chosen the size of a time step.

As boundary conditions we use constant pressure at the source Qu and a constant flow rate at sinks Se2 and Se3. The gas consumption at sink Se1 is chosen time-dependently with initially $q(x_4^b, t) = 250 \frac{\mathrm{m}^3}{\mathrm{s}}$ for $t \leq 100$ s and $q(x_4^b, t) = 300 \frac{\mathrm{m}^3}{\mathrm{s}}$ for $t \geq 105$ s and a linear increase in-between. The initial conditions are chosen stationary (Fig. 4(b)). The target functional used is $M(p,q) = \int_Q p \,\mathrm{d}x\,\mathrm{d}t$, the "Quantity of Interest" is thus the pressure measured over the whole network.

In this setting, only two models were used: the semilinear model (3) and the algebraic model (4). A reference solution was computed using the semilinear model. The equations were solved using an implicit box scheme [7]. Figure 5 shows the simulation process at times 115 s and 7075 s.

Figure 6 compares the pressure of the adaptive solution with the reference solution at sink Se1 for two different tolerances.

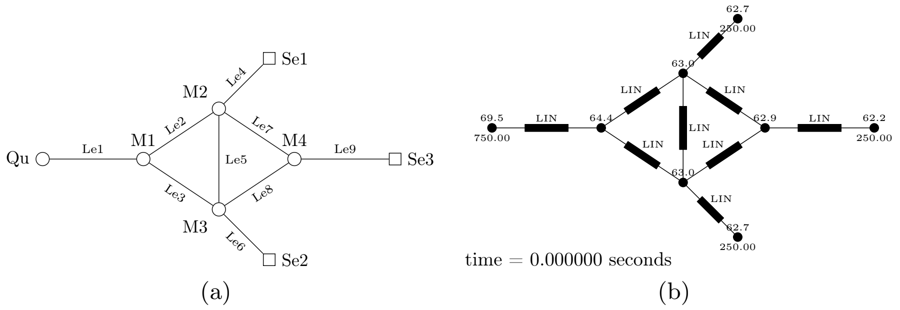


Fig. 4. (a) A small network; (b) Initial conditions; at every node the pressure p is given, at the sources and sinks additionally the flow rate q is specified below

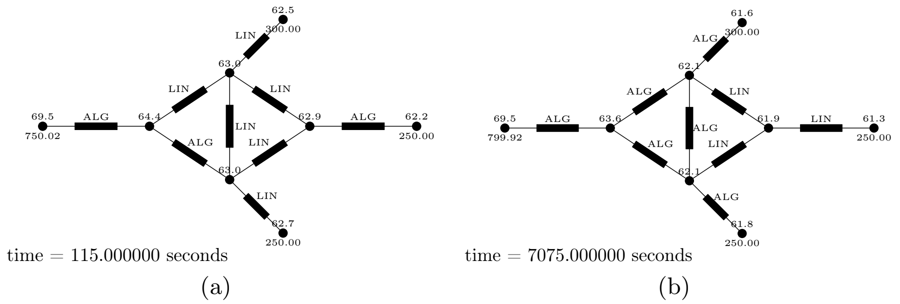


Fig. 5. Two snapshots of the simulation process using the adaptive switching strategy

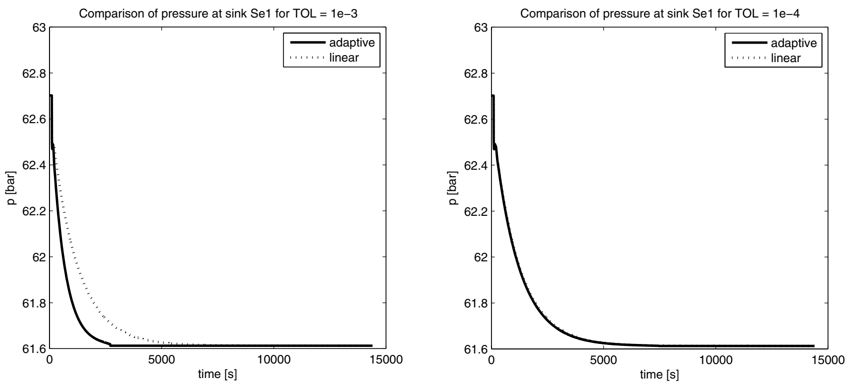


Fig. 6. Comparison of the pressure with the reference solution at sink Se1 for different values of TOL

6 Summary

We introduced a model hierarchy for the simulation of gas transport in networked pipelines. This hierarchy consists of a nonlinear and a semilinear system of hyperbolic partial differential equations and of an algebraic steady state model. We discussed coupling and boundary conditions for the wellposedness of the whole system. For the network, adjoint equations as well as adjoint coupling conditions were given that allow us to valuate the different models with respect to a quantity of interest. An algorithm was developed that switches adaptively between the three models using model error estimators deduced from the adjoint systems. The additional computational effort is approximately that of solving the original system. In the case of locally restricted dynamical effects we observed for a test network a significant reduction of complexity, while a certain accuracy is maintained. As a side result we gain an estimator for the discretization error for free.

Based on our results we want to proceed in testing the switching strategy for more complex systems including compressor stations and valves. Furthermore, an integration into an optimization framework is planned.

Acknowledgments. This paper was supported by the German Research Foundation (DFG) under the grant LA1372/5-1.

References

1. Bales, P.: Hierarchische Modellierung der Eulerschen Flussgleichungen in der Gasdynamik. Diploma thesis, TU Darmstadt (2005)
2. Bales, P., Geißler, B., Kolb, O., Lang, J., Martin, A., Morsi, A.: Comparison of Linear and Nonlinear Optimization of Transient Gas Networks. Preprint No. 2552, TU Darmstadt (2008)
3. Banda, M., Herty, M., Klar, A.: Coupling conditions for gas networks governed by the isothermal euler equations. NHM 1(2), 295–314 (2006)
4. Banda, M., Herty, M., Klar, A.: Gas flow in pipeline networks. NHM 1(1), 41–56 (2006)
5. Becker, R., Rannacher, R.: An optimal control approach to a posteriori error estimation in finite element methods. Acta numerica 10, 1–102 (2001)
6. Braack, M., Ern, A.: A posteriori control of modeling errors and discretization errors. SIAM Multiscale Model. Simul. 1(2), 221–238 (2003)
7. Kolb, O., Lang, J., Bales, P.: Adaptive linearization for the optimal control problem of gas flow in pipeline networks. Preprint No. 2553, TU Darmstadt (2008)
8. Martin, A., Möller, M., Moritz, S.: Mixed integer models for the stationary case of gas network optimization. Math. Prog. 105, 563–582 (2006)
9. Moritz, S.: A Mixed Integer Approach for the Transient Case of Gas Network Optimization. PhD thesis, TU Darmstadt (2006)
10. Sekirnjak, E.: Transiente Technische Optimierung. Concept, PSI AG (2000)
11. SIMONE, `http://www.simone.eu/simone-simonesoftware.asp`

A Hierarchical Methodology to Specify and Simulate Complex Computational Systems*

César Andrés, Carlos Molinero, and Manuel Núñez

Dept. Sistemas Informáticos y Computación.
Universidad Complutense de Madrid, Spain
{c.andres,molinero}@fdi.ucm.es, mn@sip.ucm.es

Abstract. We introduce a novel methodology to formally specify complex multi-agent systems. Our approach allows us to redefine computational problems in terms of agents that perform certain tasks. In our view, a system is formed by the combination of atomic and complex agents. Atomic agents are in charge of executing atomic tasks while complex agents reunite and summarize the properties of their underlying atomic agents. Basically, our approach consists in specifying the smaller parts of the problem as atomic agents. Each atomic agent is in charge of executing a small transformation of resources. Afterwards, the system will recombine them to form complex agents that will embrace the knowledge of several atomic agents. All agents are located on a superstructure of communication cellules created to record the hierarchy of the tasks. In order to provide a useful framework, we have developed a tool that fully implements all the stages of the methodology.

1 Introduction

Computational science embraces the concept of aiding the development of other studies in different fields through the use of new computational means. Therefore it has to create open systems that can be applied to a great extent of problems. In addition, it is relevant to take into account that the people to which computational science is directed are not, in general, computer scientists. Therefore, its easiness of use is a must. In this paper we report on a formalism that allows to solve complex problems through the use of agents. We propose a method to factorize the problem, being the first step to break down the problem into the smaller parts possible and assign an agent to each of those tasks. Then, the produced system allows to make petitions that will create other agents that, through recombination, are able to condense the information of several agents, so that they can solve a complex situation.

This paper extends and enhances our previous work presented in [1]. We have simplified some of the notations, so that the resulting formalism is much easier to use. Although we have simplified our approach, the expressive power of the framework remains the same, being able to solve the same problems that we confronted in [1].

Even though there are general purpose formalisms to formally describe complex concurrent systems (such as Process Algebras and Petri Nets) they are not suitable to

* Research supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01).

describe agents since these languages and notations do not provide specific operators to deal with the inherent characteristics of agents. However, there has been already several studies to formally describe the use of intelligent electronic agents that are nested into one another (see, for example, [2,3] for two approaches based on Petri Nets and automata, [4,5] for approaches based on process algebras, and [6,7] for approaches based on finite state machines). Most of these approaches have been created in favor of comprehensibility. Therefore they facilitate to derive and apprehend new properties. However, due to its complexity, these formalisms are not supported by suitable user-friendly tools. Thus, the specification of a system is a task that cannot be carried out by somebody that is not a real specialist in formal methods.

Our approach is able to assimilate the systems that we are interested in to a *common places* structure in which one is able to locate the rest of the structure from higher order points. If we use the subway lines as a metaphor, we only need to know the location of the different stations, but the exact location of that small fruit shop that we are trying to reach is bounded to the location of the closest metro station. Once we arrive to that particular metro station, we will check the neighborhood map so that we can find the shop; we do not need to know in advance all the local maps associated with all the stations of the network. This is how our systems will work: Once we have all the atomic agents, each time that a new complex agent, embracing the knowledge of several atomic agents, is created we will refer to this new agent when making subsequent calls to the system. In this line, we are able to *forget* how atomic actions are performed because we have a higher order element to which we can call upon. In any case, even with a complex structure, atomic agents are still the ones that execute *real* tasks.

Using another metaphor we could say that our approach produces systems that are similar to economic structures in which there exist intermediate agents that gives us the result of the transformation of resources as a final product. These agents, in a hidden way, contract the prime manufacturers that create these resource transformations. Another point in favor of our approach is that it allows us to have an unbounded growth (equivalently, subdivisions as small as needed) either by adding agents in between existing ones or by assigning new atomic agents to the system that we had before. It is important to note that the way our systems are subdivided, in so called *communication cellules*, facilitates their deployment in a distributed system in which one can obtain a perspective of variable magnitude of the global tasks. This holds as long as we keep the hierarchical structure of the ensemble.

The rest of the paper is organized as follows. In Section 2 we introduce some auxiliary notation. Section 3 represents the bulk of the paper. There we define the syntax of the proposed formalism, giving a running example of a system implemented with our tool. In Section 4 we briefly describe the technical details of the architecture of the tool developed to specify the systems. Finally in Section 5 we present our conclusions.

2 Preliminaries

In this section we introduce some notation that will be used throughout the rest of the paper. First, since users have different preferences, in order to properly design agents the first step consists in expressing these preferences. In order to extract preferences from users several mechanisms have been presented in the literature (see [8,9,10]). In this paper, preferences in a given moment will be given by a *utility function*. These functions associate a value (a utility measure) with each possible combination of resources a user could own. Alternatively, other mechanisms such as *preference relations* could be used (see e.g. [11] for conditions to transform one of the possibilities into the other).

In order to manage resources we will denote them as elements of a vector $\bar{x}$. We consider a *special* resource to record the performance of the system. The time that it takes to complete the tasks of the system will also be considered as another resource. A vector of resources is a vector of real numbers in which each number denotes the total amount of a specific resource. Along this paper we consider that n is the number of resources of the system.

Definition 1. Let $\bar{x} \in \mathbf{R}^n$ be a *vector*. We have that x_i represents the *i-th* component of $\bar{x}$. Let $\bar{x}, \bar{y} \in \mathbf{R}^n$ be two vectors. We write $\bar{x} + \bar{y}$ to denote the *addition* of $\bar{x}$ and $\bar{y}$. We say that $\bar{q}$ is the addition of $\bar{x}$ and $\bar{y}$ if $1 \leq i \leq n$ we have $q_i = x_i + y_i$.

We denote by $\bar{0} \in \mathbf{R}^n$ the vector having all the value components equal to zero. We write $\bar{x} \leq \bar{y}$ if for all $1 \leq i \leq n$ we have $x_i \leq y_i$.

A *utility function* is defined as any function $f^u : \mathbf{R}^n \rightarrow \mathbf{R}$. We denote the set of all utility functions by $\mathcal{F}$. $\square$

Intuitively, given a utility function f^u, We say that $f^u(\bar{x}) > f^u(\bar{y})$ means that $\bar{x}$ is preferred to $\bar{y}$. For instance, if we have $\bar{x} = (x_1, x_2)$ representing the first element of the resource vector the number of apples and the second element the number of oranges, $f^u(\bar{x}) = 3 \cdot x_1 + 2 \cdot x_2$, means that, for example, the agent is equally happy owning 6 apples or 9 oranges. Let us consider another agent whose utility function is $f^u(\bar{x}) = 1 \cdot x_1 + 2 \cdot x_2$. Then, both agents can make a deal if the first one gives 3 oranges in exchange of 4 apples: After the exchange both are happier. Alternatively, if x_2 represents the amount of money instead of oranges then the first agent would be a customer while the second one might be a vendor. Utility functions allow a great expressivity in preferences. For instance, $f^u(\bar{x}) = x_1 \cdot x_2$ denotes that variety is preferred. A usual assumption is that no resource is a *bad*, that is, if the amount of a resource is increased, so does the value returned by the utility function. Using a derivative expression, this property can be formally expressed as $\frac{\Delta f^u(x_1,\ldots,x_n)}{\Delta x_i} \geq 0$ for all $x_1, \ldots, x_n \in \mathbf{R}$ and $1 \leq i \leq n$. This requirement does not constrain the expressive power of utility functions, as the existence of any undesirable resource can be always expressed by considering a resource representing the *absence* of it.

Next we introduce a collection of *identifiers* to be able to univocally identify cellules, agents and paths in the system. In the next section, we will formally define these concepts.

Definition 2. Let w be a system (see Definition 8). The set of all possible systems is represented by $\mathcal{W}$. We denote by ID^{C} the set of cellule identifiers that are assigned uniquely to each of the cellules. The function $\mathbf{newIdCellule} : \mathcal{W} \rightarrow \mathrm{ID}^{\mathrm{C}}$ returns an unused identifier for the world w. We use a special identifier `nill` $\in \mathrm{ID}^{\mathrm{C}}$ to denote an empty cellule. We denote by ID^{A} the set of agent identifiers that are assigned uniquely to each of the agents belonging to the system. The function $\mathbf{newIdAgent} : \mathcal{W} \rightarrow \mathrm{ID}^{\mathrm{A}}$ returns an unused identifier for an agent. We denote by ID^{P} the set of path identifiers, that are assigned uniquely to each of the paths. The function $\mathbf{newIdPath} : \mathcal{W} \rightarrow \mathrm{ID}^{\mathrm{P}}$ returns a fresh identifier for a path. □

3 Definition of the Formalism

In this section we present our formal language to specify complete systems as well as all the agents taking part in them. The basic notion to define the behaviour of agents is a *transition*, that is, a transformation of resources carried out by a specific agent. *Atomic* and *complex* agents will both hold transitions as a way to accomplish tasks, but *only atomic agents* will actually perform the transformation of resources. A transformation of resources is represented by a tuple $\bar{z} \in \mathbf{R}^n$. Intuitively, a positive component of the tuple x_i unit of the *i-th* resource while a negative component x_j denotes that the transition consumes x_j units of the *j-th* resource.

Definition 3. A *transition* of the system is represented by the tuple $(\bar{z}, id_p)$ where $\bar{z} \in \mathbf{R}^n$ is the transformation of resources and $id_p \in \mathrm{ID}^{\mathrm{P}}$ identifies the path that is in charge of executing the transition. The set of all transitions is denoted by $\mathtt{TR}$. □

A *path* is a sequence of transitions. It is conformed of transitions, in a specific order, through concatenation. Paths allow to specify the situation where a *complex* agent has to execute several consecutive tasks.

Definition 4. A *path* is a sequence of transitions. If $tr_1, \ldots, tr_m \in \mathtt{TR}$ then $p =< tr_1, \ldots, tr_m >$ represents the path conformed by them. We have that p_i denotes the *i-th* element of the path, that is the transition tr_i. The set of all paths is denoted by $\mathcal{P}$. We can inductively define paths as follows:

– $<>\in \mathcal{P}$.
– If $tr \in \mathtt{TR}$ and $p \in \mathcal{P}$ then $tr \cdot p \in \mathcal{P}$.

Thus, paths are either empty or are conformed by adding an element to a path. □

Next we show how to represent *agents*. We can distinguish between *complex* and *atomic* agents. *Atomic* agents assume the responsibility of actually implementing tasks, and *complex* agents cluster and delegate in the ulterior ones to accomplish complex tasks and summarize the properties of the agents that are implicity inside of them.

Definition 5. An *agent* is a tuple $a = (id, ib, P)$ where $id \in \mathrm{ID}^{\mathrm{A}}$ is a unique identifier for this agent, $ib \subseteq \mathcal{M}$ is the input buffer where messages will be stored, and $P \subseteq \mathcal{P} \times \mathrm{ID}^{\mathrm{P}}$ is the set of paths defining the possible behaviours of this agent, being each path labeled with an identifier. Intuitively, the meaning of this set of paths is that this specific agent will achieve through any of this paths a similar global transformation of resources. In other words, every path takes him from the same initial state towards a similar final state, differing one from each other in the kind of transformations that they perform.

We denote by $\mathcal{A}$ the set of all agents. We define the function $\mathtt{VTr} : \mathrm{ID}^{\mathrm{P}} \longrightarrow \mathcal{P}$ as follows. Let $P = \{(<tr_1^\alpha, \dots, tr_m^\alpha>, \alpha), (<tr_1^\beta, \dots, tr_{m'}^\beta>, \beta), \dots\}$ be a set of paths. We define $\mathtt{VTr}(\alpha) = <tr_1^\alpha, \dots, tr_m^\alpha>$. We also define the function $\mathtt{VA} : \mathrm{ID}^{\mathrm{P}} \longrightarrow \mathrm{ID}^{\mathrm{A}}$ that returns the agent that performs this path. □

Let $a = (id, ib, P)$ be an agent and $P = \{(<tr_1^\alpha, \dots, tr_m^\alpha>, \alpha), (<tr_1^\beta, \dots, tr_{m'}^\beta>, \beta), \dots\}$ be the set of paths of agent a. We define the function $\mathtt{VP} : \mathrm{ID}^{\mathrm{P}} \longrightarrow \mathbf{R}^n$ using the auxiliary function $\mathtt{VPAux} : \mathcal{P} \times \mathrm{ID}^{\mathrm{A}} \longrightarrow \mathbf{R}^n$ as $\mathtt{VP}(\alpha) = \mathtt{VPAux}(\mathtt{VTr}(\alpha), \mathtt{VA}(\alpha))$ being defined as:

$$\mathtt{VPAux}(<>, id) = \bar{0}$$

$$\mathtt{VPAux}(<tr_1, tr_2, \dots, tr_n>, id) = \bar{z} + \begin{cases} \mathtt{VPAux}(<tr_2, \dots, tr_n>, id) & \text{if } tr_1 = (\bar{z}, id_p) \wedge \\ & \quad id = \mathtt{VA}(id_p) \\ \mathtt{VPAux}(\mathtt{VTr}(id_p), \mathtt{VA}(id_p)) + & \text{if } tr_1 = (\bar{z}, id_p) \wedge \\ \mathtt{VPAux}(<tr_2, \dots, tr_n>, id) & \quad id \neq \mathtt{VA}(id_p) \end{cases}$$

An agent is *atomic* if it has only one path, that path is conformed by a single transition, and the agent itself is in charge of executing the transition. Formally, $a = (id, ib, P)$ is an *atomic* agent if $\mid P \mid = 1$ and there exists $p = <tr_1, \dots, tr_m> \in P$ such that for all $1 \leq i \leq m$ if $tr_i = (\bar{z}_i, id_p)$ then $\mathtt{VA}(id_p) = id$.

During the rest of the paper we consider that agents use *messages* to communicate among them. The next definition introduces the different kinds of messages that can be sent.

Definition 6. A *message* is given by a tuple $(t, s, ob, \bar{r})$ such that $t \in \{\mathtt{BROADCAST}, \mathtt{REPLIES}, \mathtt{START\ JOB}, \mathtt{FINISHED\ JOB}\}$, denotes the nature of the message, $s \in \mathrm{ID}^{\mathrm{P}} \cup \{\mathbf{null}\}$ the path origin of the message. In some cases this path can have the value **null**. The next item $ob \in \mathrm{ID}^{\mathrm{P}} \cup \{\star\}$ is the objective of the message, it can be a specific path of an agent, or a broadcast message. The last component, $\bar{r} \in \mathbf{R}^n$ represents a transformation of resources. In the rest of this paper, we denote by $\mathcal{M}$ the set of all messages. □

Example 1. Let $id \in \mathrm{ID}^{\mathrm{A}}$ be an agent identifier, $p_1, p_2 \in \mathrm{ID}^{\mathrm{P}}$ be paths identifier, and $\bar{r}$ be a vector of resources. A message $m = (\mathtt{BROADCAST}, \mathbf{null}, \star, \bar{r})$ represents a broadcast message ($\star$) sent by a petition wanting to find an agent that accomplish the transformation induced by $\bar{r}$. If we have a message $m = (\mathtt{REPLIES}, p_1, p_2, \bar{r})$; m denotes the message from agent $\mathtt{VA}(p_1)$ that offers the path p_1, that replies to agent $\mathtt{VA}(p_2)$ to the petition of performing a certain task of the path p_2, and specifies the transformation of resources $\bar{r}$.If we have a message $m = (\mathtt{START\ JOB}, p_1, p_2, -)$, m now represents the message from agent $\mathtt{VA}(p_1)$ which is performing the path p_1 for asking to start the job to the path p_2 of the agent $\mathtt{VA}(p_2)$. Finally, if $m = (\mathtt{FINISHED\ JOB}, p_1, p_2, -)$, then m is the message from agent $\mathtt{VA}(p_1)$ to agent $\mathtt{VA}(p_2)$ to indicate that the path p_1, which is a sub-path of p_2, has just finished. □

Cellules are elements that serve as baskets of agents to reunite, organize, conglomerate and handle petitions as well as calls to the agents.

Definition 7. A *cellule* is a tuple $(\mathcal{A}, id, \text{Sons}, \text{Father}, ib)$ where

- $\mathcal{A} \subseteq \text{ID}^{\text{A}}$ is the set of agents that belong to the cellule.
- $id \in \text{ID}^{\text{C}}$ is a unique identifier for this cellule.
- $\text{Sons} \subseteq \text{ID}^{\text{C}}$ is the set of identifiers of the sons of this cellule. If $\text{Sons} = \varnothing$ then we are in a node cellule.
- $\text{Father} \in \text{ID}^{\text{C}}$ is the identifier of the cellule that is father of this cellule. If Father=`nill` then we are in the initial cellule, from which all other cellules are defined.
- $ib \subseteq \mathcal{M}$ is the input buffer where messages will be stored.

We denote by $\mathcal{C}$ the set of all cellules. □

Next, we define the whole system that contains in a tree like structure implicity defined by the father-son relationship, the cellules that conform the whole system.

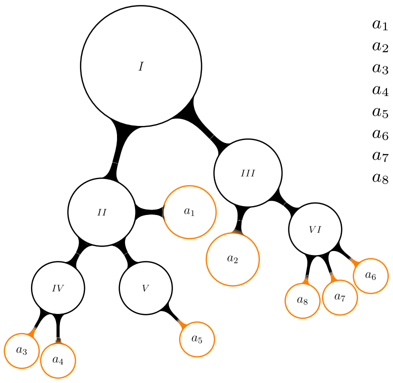
Definition 8. We say that our *system* (sometimes called *world*) is defined with a so called *origin cellule* from where the tree of cellules hang and by the vector of resources available in the system. Therefore, a *system* is a pair $w = (c, \bar{x})$ where $c \in \text{ID}^{\text{C}}$ is the origin cellule, and $\bar{x}$ is the set of resources with which we deal in this world $\bar{x} \in \mathbf{R}^n$.

We will use a running example to illustrate previously introduced concepts. In order to ease the presentation, we have simplified the real system that we have represented in our formalism. □

Example 2. Let us consider that we have the world represented in Figure 1. As we observe in the figure, we have six cellules, labeled from I to VI and eight agents distributed in them. For example, let us consider agent $a_3 = (id_3, ib_3, P_3)$. P_3 is the set of paths that this agent can perform, ib_3 represents the input buffer of this agent and id_3 is the identifier of this agent. The set of paths P_3 contains a unique pair (pair, path identifier) $P_3 = \{(<(\bar{z}_a, \kappa)>, \kappa)\}$. The path identifier is κ the first element of the pair represents the chain of transitions that compose this path. In this case the path is formed by a unique transition. This transition, $<(\bar{z}_a, \kappa)>$ represents that it is performed by the path κ of the agent $id_3 = \mathtt{VA}(\kappa)$ and the exchange of resources after performing this transition is noted by $\bar{z}$. This means that the resources of the world will change by applying $\bar{x} \leftarrow \bar{x} + \bar{z}_a$, in other words, it will generate a **formwork** unit, by wasting 50 units of **money**, 40 units of **wood**, and 20 **time** units.

For example let us suppose that agent $a_1 = (id_1, ib_1, P_1)$ has two different paths. $(<(\bar{z}_g, \kappa), (\bar{z}_g, \mu)>, \alpha)$ and $(<(\bar{z}_h, \kappa), (\bar{0}, \nu)>, \beta)$. Next we explain one of these paths. The path identified by α, has two transitions in it. The first transition, denoted by $(\bar{z}_g, \kappa)$, represents that this agent has to call to the α-*path* of agent $\mathtt{VA}(\kappa)$ to perform it, and the transformation of resources by applying this transition is $\bar{x} \leftarrow \bar{x} + \bar{z}_g$. Then, after performing this transition the resources of the world would change to $\bar{x} \leftarrow \bar{x} + \bar{z}_a + \bar{z}_g$. Let us remember that the agent $id_3 = \mathtt{VA}(\kappa)$ transformation function for the path κ is $\bar{z}_a$. Let us note that the agent performing this transition earns money by calling another agent. □

All agents that are not *atomic* are *complex*, there are two ways to create agents one is to insert an atomic agent during the creation of the system and the other is through *petitions* to the system, being the system in charge of recombining atomic and/or complex agents already embedded in the system to create a new complex agent.



$$\begin{aligned}
a_1 &= (\ id_1, ib_1, P_1\ )\\
a_2 &= (\ id_2, ib_2, P_2\ )\\
a_3 &= (\ id_3, ib_3, P_3\ )\\
a_4 &= (\ id_4, ib_4, P_4\ )\\
a_5 &= (\ id_5, ib_5, P_5\ )\\
a_6 &= (\ id_6, ib_6, P_6\ )\\
a_7 &= (\ id_7, ib_7, P_7\ )\\
a_8 &= (\ id_8, ib_8, P_8\ )
\end{aligned}$$

$$W = \{I, \bar{x}\}$$

$$P_1 = \left\{ \begin{array}{l} (<(\bar{z}_g,\kappa)\ ,(\bar{z}_g,\mu)>\ ,\alpha), \\ (<(\bar{z}_h,\kappa)\ ,(\bar{0},\nu)>\ \ ,\beta) \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{l} (<(\bar{z}_g,\varpi)\ ,(\bar{0},\varsigma)>\ ,\gamma), \\ (<(\bar{0},\varpi))\ ,(\bar{0},\tau)>\ ,\delta), \\ (<(\bar{0},\upsilon)\ \ \ ,(\bar{0},\varsigma)>\ ,\epsilon), \\ (<(\bar{z}_h,\upsilon)\ ,(\bar{0},\tau)>\ ,\iota) \end{array} \right\}$$

$$P_3 = \{(<(\bar{z}_a,\kappa)>,\kappa)\}$$

$$P_4 = \{(<(\bar{z}_b,\mu)>,\mu)\}$$

$$P_5 = \{(<(\bar{z}_c,\nu)>,\nu)\}$$

$$P_6 = \{(<(\bar{z}_d,\varpi)>,\varpi)\}$$

$$P_7 = \left\{ \begin{array}{l} (<(\bar{z}_{e1},\varsigma)>,\ \varsigma), \\ (<(\bar{z}_{e2},\tau)>,\ \tau) \end{array} \right\}$$

$$P_8 = \{(<(\bar{z}_f,(id_8,\upsilon))>,\upsilon)\}$$

$$\begin{array}{lllllllllll}
\bar{z}_a &= [& -50, & 0 & 0 & -40, & 0, & 0, & 0, & 1, & -20\ ] \\
\bar{z}_b &= [& -100, & -300, & 0, & 0, & 1, & 0, & 0, & -1,, & -30\ ] \\
\bar{z}_c &= [& -80, & -450, & 0, & 0, & 1, & 0, & 0, & -1, & -40\ ] \\
\bar{z}_d &= [& -200, & 0, & -300, & 0, & 1, & 0, & 0, & 0, & -20\ ] \\
\bar{z}_{e1} &= [& -50, & -10, & 0, & 0, & 0, & -20, & ,1, & 0, & -30\ ] \\
\bar{z}_{e2} &= [& -55, & -9, & 0, & 0, & 0, & -23, & 1, & 0, & -15\ ] \\
\bar{z}_f &= [& -250, & 0, & -250, & 0, & 1, & 0, & 0, & 0, & -30,\ ] \\
\bar{z}_g &= [& -25, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & -20\ ] \\
\bar{z}_h &= [& -30, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & -10\ ]
\end{array}$$

Position in the resource tuple of the world	Semantic
1	**money**
2	**concrete**
3	**steel**
4	**wood**
5	**structure**
6	**bricks**
7	**wall**
8	**formwork**
9	**time**

Fig. 1. Representation of a world

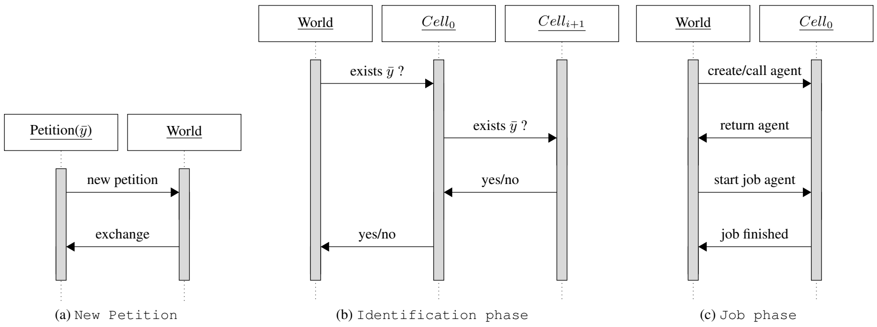


Fig. 2. Schematic diagrams of world behaviour

Definition 9. We say that a *petition* is a tuple $pet = (f^u, \bar{y}, \bar{o})$ where $f^u \in \mathcal{F}$ is a utility function, $\bar{y} \in \mathbf{R}^n$ is the vector of resources that is added to the resources already existing in the world, and $\bar{o} \in \mathbf{R}^n$ is the objective of the transitions, that is, the vector of resources that we expect to have after performing the petition. □

Intuitively, if we have a petition $pet = (f^u, \bar{y}, \bar{o})$ is a petition, and $a = (id, ib, P)$ is the agent that has created the petition, if $\exists\ p \in \texttt{TR}$ such that exists $(p, id_p) \in P$: $\texttt{VP}(id_p) + \bar{x} + \bar{y} \geq \bar{o}$.

We will explain the main messages by applying a petition (see a graphical representation in Figure 2). Let us consider a petition $pet = (f^u, \bar{y}, \bar{o})$. The first component of pet contains the initial resources, $\bar{y} = [1000, 500, 500, 100, 0, 100, 0, 0, 500]$, the second one is a utility function (in this case $f^u = 10 \cdot x_1 + 5 \cdot x_9$), and the third element is the objective tuple of resources $\bar{o} = [0, 0, 0, 0, 1, 0, 1, 0, 0]$.

The first diagram of Figure 2 denotes that $pet = (f^u, \bar{y}, \bar{o})$ is inserted in the world $w = (I, \bar{x})$. When a new petition is inserted in the world, the resources of the petition are added to the existing vector of resources. After this initial stage, the world "asks" to its structure of cellules if there are any agent(s) which can achieve the objective function $\bar{o}$.

4 Implementation

In this section we present our tool that facilitates the task of representing the different components of our framework. First, we are going to enumerate some of the technical requirements of the tool. Next, we will comment on some relevant parts of the implementation, and we will show how the example can be represented.

The tool has been developed using J2EE Technology (Java, JDK 1.5, EJB) and the Netbeans software. It makes usage of MVC architecture, to enable ease of maintenance, and uses session facade and proxy design patterns. It also uses Java Swing components in order to develop Graphical User Interfaces(GUI).

The tool offers four different ways to create systems. The first one is by using an input XML-formatted file which contains all the description data of the system. Another way to introduce a model is by using Java Database Connectivity (JDBC). JDBC is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. The third way to input a system is by using the editor included in the GUI. The user may customize the editor and make sure the Editor Presentations command group is checked under the Commands tab. The last way to create systems is by loading serializable models saved previously. Serializability of a class is enabled by the class implementing the java.io.Serializable interface. By using an MVC design, the system can easily make it. When a model is in the tool, it can be also saved in the same ways, by using an XML formatted file, in a database, and by serialization of the model.

For the representation of the world, cellules, and agents we have used *threads*. A java.lang. Thread object maintains bookkeeping and control for this activity. In fact, by representing each of the components by using threads we let the system represent a more realistic world. For example agent a_1 can be waiting until agent a_3 has finished its task, while the world continues receiving petitions, and the cellules continue sending messages between the agents. The tool also introduces priority among the threads. Each Thread has a priority, ranging between 1 and 10. Priorities have no other impact on semantics or correctness. In particular, priority manipulations cannot be used as a substitute for locking. Priorities can be used only to express the relative importance or urgency of different threads, being these priority indications useful to take into account when there is heavy competition among threads waiting to be executed. For example the initial cellule normally has bigger priority than other cellules; the reason is that management of petitions and the data traffic flow is mainly done through this cellule.

Another important task in a concurrency scheme is the management of shared memory, being the buffers implemented as circular buffers using a single, fix size. Circular buffers are also used for data transfer between processes. The tool uses monitors in theses buffers to synchronize accessing threads. Conceptually, a monitor is a class whose data members are private and whose member functions are implicitly executed with mutual exclusion. In addition, monitors may define waiting conditions that can be used inside the monitor to synchronize the members functions.

5 Conclusions and Future Work

The work presented in this paper provides a useful framework for the developing of complex computational systems. With the presentation of the tool we offer the possibility of use to specialists from different fields, not being constrained its applicability to the field of computing technology and therefore being useful as a computational science tool.

The work will be extended to allow testing and checking of conformance of the system implemented with it. Other future line of work will be to develop the structure of the cellules in an AVL tree-like structure, so that the message flow will be re-equilibrated. Another possible continuation of the work will be to specify and implement the subdivision of cellules when the number of agents that they withholds surpasses a limit, this allows not to surcharge a specific computational resource in which the cellule may be implemented, like a specific processor, and derive its work flow to another resource.

Finally, we are currently working on a complex application of our methodology to the construction world.

References

1. Andrés, C., Molinero, C., Núñez, M.: A formal methodology to specify hierarchical agent-based systems. In: 4th Int. Conf. on Signal-Image Technology & Internet-based Systems, SITIS 2008, pp. 169–176. IEEE Computer Society Press, Los Alamitos (2008)
2. Lomazova, I.: Communities of interacting automata for modelling distributed systems with dynamic structure. Fundamenta Informaticae 60(1-4), 225–235 (2004)
3. Lomazova, I.A.: Nested Petri Nets for Adaptive Process Modeling. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 460–474. Springer, Heidelberg (2008)
4. Núñez, M., Rodríguez, I.: PAMR: A process algebra for the management of resources in concurrent systems. In: 21st IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE 2001, pp. 169–185. Kluwer Academic Publishers, Dordrecht (2001)
5. Núñez, M., Rodríguez, I., Rubio, F.: Formal specification of multi-agent e-barter systems. Science of Computer Programming 57(2), 187–216 (2005)
6. Núñez, M., Rodríguez, I., Rubio, F.: Specification and testing of autonomous agents in e-commerce systems. Software Testing, Verification and Reliability 15(4), 211–233 (2005)
7. Merayo, M., Núñez, M., Rodríguez, I.: Formal specification of multi-agent systems by using EUSMs. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 318–333. Springer, Heidelberg (2007)
8. Dastani, M., Jacobs, N., Jonker, C., Treur, J.: Modelling user preferences and mediating agents in electronic commerce. In: Sierra, C., Dignum, F.P.M. (eds.) AgentLink 2000. LNCS, vol. 1991, pp. 163–193. Springer, Heidelberg (2001)
9. Geisler, B., Ha, V., Haddawy, P.: Modeling user preferences via theory refinement. In: 5th Int. Conf. on Intelligent User Interfaces, IUI 2001, pp. 87–90. ACM Press, New York (2001)
10. Ha, V., Haddawy, P.: Similarity of personal preferences: Theoretical foundations and empirical analysis. Artificial Intelligence 146(2), 149–173 (2003)
11. Mas-Colell, A., Whinston, M., Green, J.: Microeconomic Theory. Oxford University Press, Oxford (1995)

GRID Resource Searching on the GridSim Simulator

Antonia Gallardo[1], Luis Díaz de Cerio[2], Roque Messeguer[1], Andreu Pere Isern-Deyà[3], and Kana Sanjeevan[1]

[1] Departament de Arquitectura de Computadors, Universitat Politécnica de Catalunya, Avda. del Canal Olímpic s/n, 08860 Castelldefels, Barcelona, Spain
[2] Departamento de Automática y Computación, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006 Pamplona, Spain
[3] Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears, Ctra. de Valldemossa, km. 7.5, 07122, Palma de Mallorca, Illes Balears, Spain
{agallard,meseguer,sanji}@ac.upc.edu, luismanuel.diazdecerio@unavarra.es, reupere.isern@uib.es

Abstract. Nowadays, the Grid is the focus of multiple researches. Our work is centered on Resource Management for Grids as it is an opened and current research area. Decentralized, scalable and efficient resource search algorithms are one of the key issues for resource management in large Grid systems. Resource search is required in order to allocate applications and data efficiently and to maintain the quality of service at runtime, just to mention some examples. In this work, we propose a scheme that presents essential characteristics for self-configuring search and is able to handle dynamic resources, such as memory capacity. Our approach consists on a hypercube topology connecting the nodes and a scalable and self-configuring search procedure. The algorithm improves the probability of reaching the alive nodes in the Grid even in the presence of non-alive ones (inaccessible, crashed or heavy loaded nodes). In this paper, after the theory's description, we present some results obtained by running our search protocol on the GridSim simulator. We have evaluated 6 different metrics performing several resources searches and we show the arithmetic media for each measure.

Keywords: GridSim, Hypercube, Self-Configuring, Search Algorithms.

1 Introduction

The Grid includes a large number of dynamic and heterogeneous resources that are geographically distributed. Its main objective is to use the available resources provided by administrative domains, also named Virtual Organizations (VO) [1]. Grid Resource Management in general and Resource Search in particular is an opened research topic. This challenge plays a fundamental role, for example, allowing the system to allocate grid-enabled applications and data efficiently and to maintain the quality of service of the applications at runtime. In most of the current grids Resource Management solutions are based on centralized or hierarchical structures that are not appropriate for large systems because they are not scalable enough. By the other hand, several scalable and decentralized Peer-to-Peer (P2P) resource searching algorithms have been proposed for Grid systems. Nevertheless, P2P searching techniques were designed for non-dynamic content as files, and the Grid requires addressing dynamic resource data such as available memory, processor load, etc., for which, the P2P search is not suitable. Besides, as pointed out by Ian Foster and al. [2], "it is necessary to address failure, using scalable self-configuring protocols so to have a worldwide computer within which access to resources can be negotiated as and when needed". Motivated by this open research area, we present a scalable and decentralized architecture that allows the search of distributed resources preserving the VO's autonomy. The architecture is based on an overlay network with hypercube topology that interconnects nodes and each node represents a VO. We also present a self-configuring resource search algorithm that is able to adapt himself to environments where some nodes might be non-alive (crashed, inaccessible, high-loaded,…) when a resource is queried. Finally, we present some results obtained by running our search protocol on the GridSim simulator. We have evaluated 6 different metrics performing several resources searches and we show the arithmetic media for each measure.

One of the challenges of Grid Resource Searching is to be resilient in the presence of node failures. This resilience has different aspects: *static resilience* and *routing recovery*. As the present work is focused on the resource discovery algorithm this paper only addresses *static resilience* [3], that is, how well our approach can locate required resources before routing tables are updated by the routing recovery algorithm in order to remove non-alive nodes from the overlay. The other issue, *routing recovery*, deals with the fact that when failures occur, the routing tables are depleted in the remaining nodes. Routing recovery is not addressed in this paper, as this issue is related to the building and maintaining of the overlay topology.

The rest of the paper is organized as follows: In Section 2 we present an overview of our network architecture. Section 3 describes our resource search procedure. An overview of related work is presented in Section 4. In Section 5, the performance of the algorithm is shown. Conclusions and our future work can be found in Section 6.

2 The Hypercube Overlay Architecture

In actual Grid environments, the administrative domain (VOs), do not join or leave the Grid continually but occasionally. Most of the current VOs that form a Grid have powerful servers within their high performance local area networks and maybe they are interconnected by very high speed core networks. The servers seldom fail and they join and leave the system infrequently. Therefore the Grid Resource System can be organized in a stable and regular topology with low-diameter configurations, efficient searching and routing algorithms that address node failures. Each VO belonging to our environment, named HGrid, provides available resources and makes them accessible through software entities named Grid Information System (GIS) [4].

In HGrid, the interconnections between nodes have the topology of a hypercube. An n-dimensional hypercube (H_n) has $N = 2^n$ nodes, where each node represents a GIS and they have an identifier from 0 to 2^n-1. Two nodes are neighbors in the m-th dimension if the binary representations of their identifiers differ exactly by the m-th bit. Then, in a complete hypercube H_n, each node has exactly n neighbors. Fig. 1

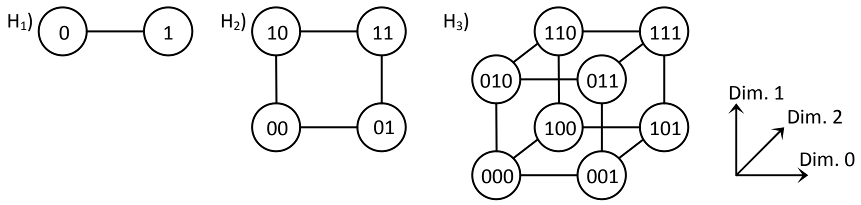


Fig. 1. H_1) The architecture for the interaction among 2 nodes, a one-dimensional hypercube, H_2) 4 nodes, a two-dimensional hypercube and H_3) 8 nodes, three-dimensional hypercube

illustrates the architecture for 2, 4 and 8 nodes respectively. An overlay network with a hypercube topology connecting each GIS in a grid environment allows each VO to contribute their resources while assuring their autonomous management. The resources offered by a VO can join or leave the system at any time updating its own GIS. Every GIS keeps a small routing table of only n entries ($n = \log_2 N$) corresponding to their n neighbors. Also, each GIS has the responsibility to verify which of its neighbors are still alive. The term alive is applied to a neighbor node that can be reached across the network. Then, a node that has crashed, is inaccessible or is heavily loaded by traffic is considered a non-alive node.

3 Resource Search Using HGRID

We present a scalable self-configuring resource search algorithm that is able to adapt him automatically to dynamic environments. Inside HGrid, the approach named Algorithm-H, is performed to search a resource/s requested by a client. It is a self-configuring protocol by adapting itself when some nodes are in a non-alive state (inaccessible, crashed or heavy loaded nodes). As we said before, we do not address the building and maintaining of the hypercube topology when a node joins or leaves the overlay. Changes in the overlay network make the routing tables be re-mapped and this does result in some overhead. However, some of the results published regarding the scalability of hypercube overlays used in other environments seem to address this challenge and offer an adequate solution [5].

We assume that in Grid environments the VOs joins or leaves the Grid occasionally, so the overhead of building and maintaining the hypercube overlay is smaller than in an extremely transient environment - where a significant fraction of the nodes are joining or leaving at any time. Since an incomplete hypercube could be re-built as a complete hypercube where the void spaces are completed by replicating some of the GIS, we have considered the number (N) of GIS is always a power of 2, so all the nodes have n-neighbors - where n is the dimensionality of the hypercube. Finally, in our proposal it is possible to initiate a search request from any of the GIS nodes alive and to propagate the request to the rest of nodes. But for generality reasons, all the examples used from now on assume that GIS 0 is the start node.

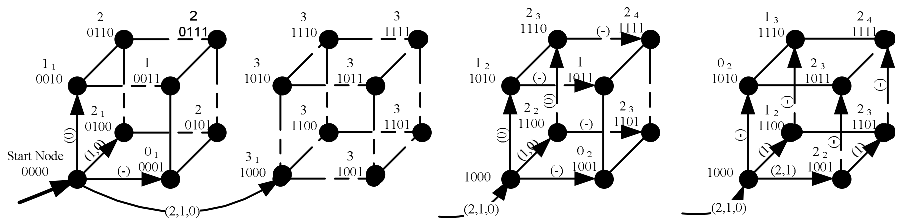


Fig. 2. The Search Procedure in a H_4 hypercube. a) Decomposing a hypercube H_4 into four hypercubes. b) Searching in the 3-dimensional hypercube marked with 3 in a). Each reached node applies the same procedure based on the received vector. c) A possible reorganization of the case showed in b). Before sending the query, every reached node can reorganize its hypercube of several forms.

3.1 The Search Procedure in a H_4

The search starts when a client connects to a GIS node and it requests a resource/s. If the start node does not have the resource/s requested, then it starts a search. Next, we show the search in a 4-dimensional hypercube:

1) The start node 0, in decimal notation, of a 4-dimensional hypercube is connected across its 4 neighbor nodes to 4 different hypercubes. These 4 hypercubes are a hypercube of 2^0 nodes in dimension 0, one of 2^1 nodes in dimension 1, one of 2^2 nodes in dimension 2 and finally, one of 2^3 nodes in dimension 3. In Fig. 2.a) we have marked these hypercubes with 0, 1, 2 and 3, respectively. Notice that these 4 hypercubes contain all the nodes of the overlay except the initial node.
2) When the start node does not have the requested resource/s, it starts a search by sending the query to each of its neighbors. It is the 1st step of 4. In this step, the nodes reached are marked by the sub-index 1 in Fig. 2.a) as 0_1, 1_1, 2_1 and 3_1.
3) Each reached node receives a first vector along with the request. This vector indicates the dimensions through which the request will be send in case of it cannot be satisfied by the reached node.
4) Each reached node applies the same procedure described in previous steps - 1), 2) and 3) - for its neighbours formed by the received dimensions. We show in Fig. 2.b) the 3-dimensional hypercube marked with 3 in Fig. 2.a). Each GIS node has an identifier in H_4, the decomposed hypercube dimension of which it forms a part, the vector received and the search algorithm step when it is requested. In Fig. 2.b), the node 1010 forms a part of *a* 1-dimensional decomposed hypercube, the node receives the vector 0 and it is reached in the step 2 of the search procedure.
5) Before sending the query, each reached node can reorganize its hypercube of several forms. We show in Fig. 2.c) a possible reorganization of the case illustrated in Fig. 2.b) for the 3-dimensional hypercube marked with 3 in Fig. 2.a). In Fig. 2.c), the node 1010 forms a part of a 0-dimensional decomposed hypercube, the node receives an empty vector (marked by (-)) and it is reached in the step *2* of the search algorithm. When a node receives an empty vector, it does not propagate this vector anymore. Notice that the nodes covered in 2c) are the same nodes those reached in 2.b), nevertheless, the node 1010 propagates the

query to nobody in Fig. 2.c) and propagates to one neighbour (the node 1011) in Fig. 2.b).

6) What does happen if node 1011 has the required resource(s) and node 1010 is a non-alive node? In this case, if node 1000 knows that node 1010 is not alive (maybe it is crashed), it reorganizes its hypercube as in Fig. 2.c) and the node 1011 is not reached across the non-alive node by trying to be queried to another.
7) The search procedure reorders the vector received with the resource query in order to send the query at maximum number of nodes that was possible.
8) Each reached node receives a second vector, too. The search procedure tries to requested GIS nodes through nodes in an alive state, by avoiding the non-alive ones. This second vector is not illustrates in Fig. 2, but it is shown in Fig. 3.

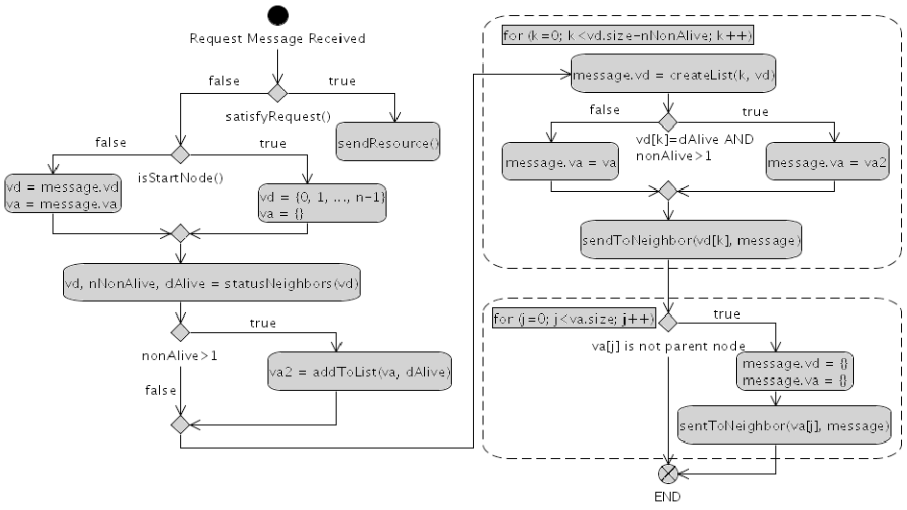


Fig. 3. Flow Diagram of the Algorithm-H

Propagating the requests in this way, the effect of non-alive nodes is reduced. Making the arrangement in the vector received, non-alive nodes would propagate the request to fewer neighbors than alive ones. Consequently, the algorithm tries to isolate the nodes that are in a non-alive state so that they become leaf nodes - if it is possible. For the start node and for each node that receives a non-empty vector, if only one neighbor node of those to which the search must be propagated is in a non-alive state, the total number of nodes reached at the algorithm last step is not affected. Related to the alive nodes unreachable due to the fact its non-alive parent's node, Algorithm-H tries to reach them using the v_a list. In Fig.3, the flow diagram is illustrated.

3.2 A Complete Example Using Algorithm-H

Fig. 4 illustrates a complete example. We transform the hypercube representation to that of a *tree-like* structure in order to illustrate better our search procedure (notice, some *child* nodes could appear more than once during subsequent time steps).

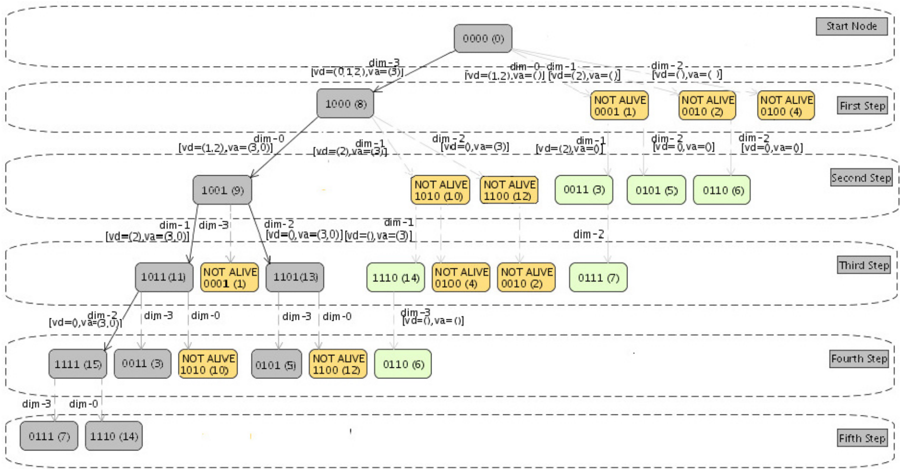


Fig. 4. Algorithm-H: A request of a resource/s P started at node 0000 in a 4-dimensional hypercube (A complete example)

A request for service P starts at node 0000 in a four-dimensional hypercube. We assume that none of the nodes has the service requested (note that this is the worst case). In the example, the value of the list vd at the start node is {0, 1, 2, 3} and the ordering after calling the statusNeighbors() function is {3, 0, 1, 2}. In this case 0, 1 and 2 are located at the last three positions of vd = {3, 0, 1, 2} because we assume that neighbors in dimensions 0 (0001), 1 (0010), 2 (0100) and are non-alive nodes. The neighbor in dimension 3 (1000) is the last alive node - the only one in this case.

In the first step, the start node's neighbor in dimension 3 (1000) receives the service request *P*, the list v_d = {0, 1, 2} and v_a = {3} since it is the last alive neighbor. The algorithm tries to reach nodes 0010, 0101 and 0110 (whose parent nodes are non-alive) by sending the list v_a = {3} to node 0010 to be used in the third stage.

In the second step, looking at node 1001, the message composed of the resource request *P* along with the lists v_d = {1, 2} and v_a = {3} is received. If the node is unable to satisfy the request - *processRequest()* returns *false* -, v_d is not sorted because its neighbor in dimension 1 (1011) and its neighbor in dimension 2 (1101) are alive nodes. Although v_a = {0} is received by the node 1001, it does not propagate the message to its neighbor in dimension 0 (1000) because it is its parent node.

In the third phase, looking at node 1011, the message composed of the resource request *P* along with the lists v_d = {2} and v_a = {3,0} is received. If *processRequest()* returns *false*, its neighbor in dimension 2 (1111) receives *P*, v_d = { } and v_a = {3,0} and nodes (0111) and (1110) receive *P*, v_d = { } and v_a = { }, due to the list v_a = {3,0}.

In the fourth stage, the resource request *P* is received from node 1011 (0011 is the neighbor in dimension v_a [0] = 3) to the node 0011. Notice that 0011 was not reached in the 2nd step due to the non-alive node 0001 but it is reached now.

In five steps almost all of the alive nodes inside the four-dimensional hypercube are visited (except node 0110) even when three neighbors of the start node (0001, 0010 and 0100) and two more nodes (1010 and 1100) are presumed to be non-alive.

4 Related Work

The Globus Toolkit's Monitoring and Discovery System (MDS) defines and implements mechanisms for resource discovery and monitoring in Grid environments [6]. Motivated by these issues, recently there have been several studies using the P2P model to build a decentralized architecture of VOs. Most of them adopt Distributed Hash Tables (DHTs) and a few of them introduce unstructured P2P topologies. All these studies indicate that some P2P models could help to overcome the challenges posed by the dynamic environment in Grids. Adriana Iamnitchi and Ian Foster [7] suggest a decentralized architecture similar to the Gnutella P2P system. This approach is not able to guarantee that some required information that exists in the system can be found even when the system has no failures. Moreover, a peer could be often reached several times by the same query. Our approach assures that nodes are reached only once. In the absence of failures all nodes in the system are reached and if some failures occur the most nodes are reached.

DHT based systems handle unexpected node failure through redundancy in the network and some of them also do node asynchronous lookups periodically to compensate for disappeared nodes – for example, Kadmelia [8]. To enable efficient searches a DHT needs to have the data-item distributed. Our approach does not require distributing the data-items but each request sends from 0 to *N*-1 messages.

Keeping the state of highly dynamic data-items updated - such as available memory or CPU processing - requires sending a large amount of messages in DHTs approaches– as in SWORD [9]. In DHTs, the data-item that belong to a VO, are geographically distributed and, an updating operation reaches O(log *N*) of distributed peers before inserting the data item (where *N* is the total number of peers). Our approach takes advantage of the high quality network available inside a VO (probably a LAN): updates are more efficient inside a VO than among distributed VOs. Furthermore, HGrid is more efficient in the insert phase as it is performed locally.

A node in the traditional DHT has no control over the distribution of its data items, and the number of data items belonging to others that it has to store. DGRID [10], a model for supporting GIS over the DHT Chord, maintains the resource information in the originating VO by increasing the total number of DHT nodes. Unlike traditional DHTs, DGRID is by design resilient to node failures without the need to replicate data items. The meaning of resilient to node failures is defined as the ability to locate existing resources whose originating VO is still alive. The approach presented in this paper has the same resiliency to node failures as in DGRID and also guarantees that any data item can be located in O(log N) overlay hops.

In HGrid, changes in the overlay network when a grid node joins or leaves the system do not cause resource information (data-items) to be remapped, whereas in traditional DHTs, it causes both routing tables and data-items to be remapped.

Recently, an unstructured topology based on hypercube has been proposed for use on Data Grids [11]. The nodes in this work contain pointers to shared data. Data Grids need to improve locality among distributed data - which are stored as pointers in the overlay nodes. In order to improve the locality of data, the paper imposes a hypercube

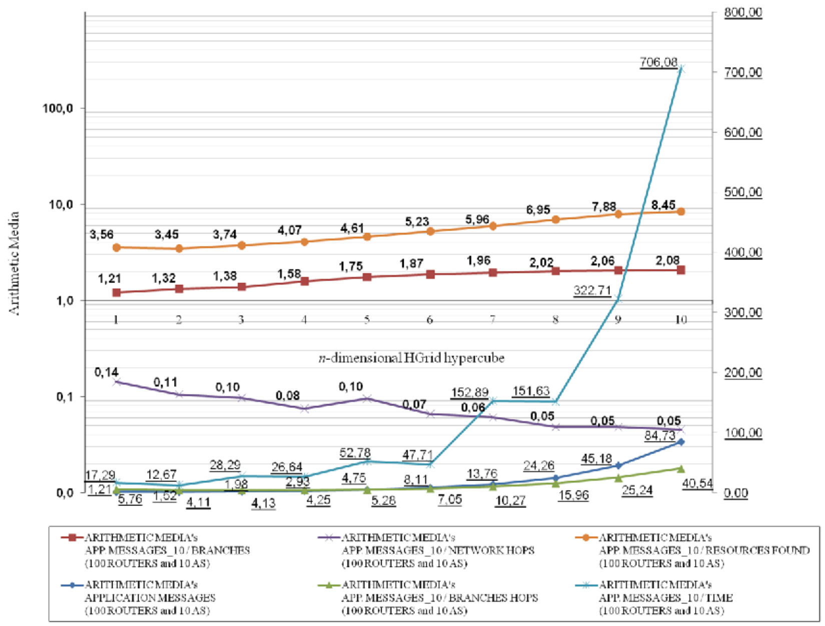


Fig. 5. HGrid and Algorithm-H: For each parameter, once set the (P_{GIS}, P_{REC}) values, we performed 20 resources searches and we obtained the arithmetic media

Characteristics	Computer-1	Computer-2
CPU	P4 2.6GHz	C2D 1.6GHz
RAM Memory	1.5 Gb	2 Gb
Hard Disk	1 x 120GB + 1 x 160GB	1 x 160GB

Fig. 6. The Computers used on our experiments with HGrid and Algorithm-H's searches

GIS's topology - named DGIS. After this, it proposes a transposition algorithm in order to optimize the overlay network's topology according to the access statistics between peers - that is, to improve the data locality. However, the algorithm showed does not address non-alive nodes and failures.

Finally, we have evaluated HGrid and Algorithm-H in previous works. For example, in [12], we show the percentage of average of failed paths across different search algorithms (HaoRen et al.'s algorithm - a non fault-tolerant algorithm described in [11] - and Algorithm-P - our previous Algorithm-H fault-tolerant protocol described in [13].

5 Performance Evaluation

In Fig.5, we tested the *static resilience* of Algorithm-H inside 1 to 10-dimensional HGrid overlays. The experiments run in 2 computers described in Fig.6. All HGrid's GIS had a P_{GIS} probability - probability of failure - and a probability P_{REC} - probability to have the resource/s requested when the queried arrives to a GIS. P_{GIS} can be seen as the percentage of non-alive nodes that can be in and P_{REC} as the percentage of resources requested presents in the HGrid hypercube when a search is performed. The BRITE [14] and the GridSim [15] was used to create automatically the IP Internet topology and the HGrid, respectively. Given a (P_{GIS}, P_{REC}) we started 20 requests for service *P* at and we calculated the arithmetic media for each measure.

We have evaluated the following metrics for each resource/s search in HGrid using Algorithm-H: a) the total number of messages sent in the hypercube overlay, b) number of branches opened where a branch is each of the searching ways in which it is divided the requested message initiated by the client, c) for each opened branch, the number of application hops, d) for each branch established, the network hops performed through routers in the overlay, e) by simulation, the search estimated time, where the time was set as the difference between the time when the request resource arrives at the last GIS and the moment in which the user initiates his search and the number of the resources found during the search in HGrid. In Fig.5 we show the inter-relation between the metrics b) - e) and the total messages application sent - a).

To summarize, our results confirm that the static resiliency of the algorithm shown is very efficient for current non-extremely transient Grid environments. It offers high lookup guarantees and it seems to be scalable with the number of nodes.

6 Conclusions and Future Work

The present approach allows the search of geographically distributed resources while preserving the autonomy of each individual VO. Unlike traditional approaches based on DHTs our scheme is suitable for efficiently handling dynamic attributes such as memory capacity without generating overhead across distributed nodes. HGrid using Algorithm-H is able to adapt him automatically to environments where nodes could be heavily loaded or even crashed, without requiring any node to have the global state information. We refer to this property as self-configuring. In the absence of non-alive, if some node present in the overlay is able to satisfy the request this node will be found in few steps (less than the hypercube dimension). Therefore the proposed scheme offers lookup guarantees in the absence of faulty nodes. If non-alive nodes are present, the algorithm also offers lookup guarantees in some cases.

To conclude, there are interesting areas that have opened up as a result of this work as comparing several metrics between the present approach and some actual DHT-approaches, to evaluate if the router layer affects to the overall resource searches, new search algorithms - design and simulation -, the incorporation of topology maintenance protocols and performing studies related to other topologies such as ring or mesh. Finally, deploy the algorithms into a real GRID is a good form to end this work, basically, a desirable final step.

Acknowledgements

This work was supported by the Ministry of Science and Technology of Spain and the European Union (TIN2007-68050-C03-01 and TIN2007-68050-C03-02).

References

1. Nabrzyski, J., Schopf, J.M., Weglarz, J.: Grid Resource Management. State of the Art and future Trends. Kluwer Publishing, Academic publishers (2004)
2. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735. Springer, Heidelberg (2003)
3. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The Impact of DHT Routing Geometry on Resilience and Proximity. Applications, Technologies, Architectures and Protocols for Computer Communications, 381–394 (2003)
4. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience 14(13-15), 1175–1220 (2002)
5. Liebeherr, J., Beam, T.K.: HyperCast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology. In: Rizzo, L., Fdida, S. (eds.) NGC 1999. LNCS, vol. 1736, pp. 72–89. Springer, Heidelberg (1999)
6. The Globus Toolkit, `http://www.globus.org/` (last access 15/02/2009)
7. Iamnitchi, A., Foster, I.: On fully decentralized resource discovery in grid environments. In: Lee, C.A. (ed.) Grid 2001. LNCS, vol. 2242, pp. 51–62. Springer, Heidelberg (2001)
8. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
9. Oppenheimer, D., Albrecht, J., Patterson, D., Vahdat, A.: Design and Implementation Tradeoffs for Wide-Area Resource Discovery. In: 14th IEEE Symposium on High Performance Distributed Computing, Research Triangle Park, NC USA (HPDC 2005) (2005)
10. March, V., Teo, Y.M., Wang, X.: DGRID A DHT-Based Resource Indexing and Discovery Scheme for Computational Grids. In: 5th Australasian Symposium on Grid Computing and e-Research (AusGrid 2007), pp. 41–48 (2007)
11. Ren, H., Wang, Z., Liu, Z.: A Hyper-cube based P2P Information Service for Data Grid. In: Conference on Grid and Cooperative Computing (GCC 2006), pp. 508–513 (2006)
12. Gallardo, A., Díaz de Cerio, L., Sanjeevan, K.: Scalable Self-Configuring Resource Discovery for Grids. In: V Brazilian Workshop on Grid Computing and Applications (WCGA 2007) (2007)
13. Gallardo, A., Díaz de Cerio, L., Sanjeevan, K., Bona, L.C.E.: HGRID: An Adaptive Grid Resource Discovery. In: 2nd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008) (2008)
14. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: Universal Topology Generator from a User's Perspective, pp. 1–47 (2001), `http://www.cs.bu.edu/brite/` (last access 15/02/2009)
15. Buyya, R., Ranjan, R., Broberg, J., Dias de Assuncao, M.: GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing, `http://www.gridbus.org/gridsim/` (last access 15/02/2009)

Evaluation of Different BDD Libraries to Extract Concepts in FCA – Perspectives and Limitations

Andrei Rimsa, Luis E. Zárate, and Mark A.J. Song

Department of Computer Science, Applied Computational Intelligence Laboratory
Pontifical Catholic University of Minas Gerais - Brazil
rimsa@live.com, {zarate,song}@pucminas.br

Abstract. This paper presents evaluation of different types of Binary Decision Diagrams (BDDs) applied to Formal Concept Analysis (FCA). The aim is to increase the FCA capability to handle large formal contexts and perform faster operations over different types of this data structure. The main idea is to represent formal context using BDDs for later extraction of the set of all formal concepts from this implicit representation. A comparison of a concept extraction algorithm using contexts implemented as table and BDD are presented. BDD is evaluated over two different implementation libraries, BuDDy and CUDD. A ZBDDs (Zero-Supressed BDDs) version of the concepts extraction algorithm is also provided. BDD has been evaluated based on several types of randomly generated synthetic contexts with large amounts of objects. Contexts are evaluated according to the computational time complexity required to build and extract the set of all concepts from it. In this work, it is shown that BDD could be used to deal with large formal contexts especially when those have few attributes and many objects. To overcome the limitations of having contexts with fewer attributes, one could consider vertical partitions of the context to be used with distributed FCA algorithms based on BDD.

Keywords: Formal Concept Analysis, Formal Context, Formal Concept, Binary Decision Diagrams, Zero-Supressed Binary Decision Diagrams.

1 Introduction

At the International Conference on Formal Concept Analysis in Dresden (ICFCA 2006) an open problem of "Handling large contexts" was pointed out and as an example was cited the challenge of "how to calculate/generate all concepts of a large context" (e.g. 120,000 x 70,000 objects attributes). In these cases, traditional FCA algorithms have high computational cost and demand high execution times, making the extraction of all concepts infeasible for larger contexts.

One possible solution to deal with the problem of handling large formal contexts is to apply a distributed solution for the processing of contexts. Partial concepts are obtained for later merging through a specific operator to find the final set of concepts. Several authors have presented formal proposals and mathematical formalisms for distributed application of FCA, as can be seen in [1-3].

It is clear the potential of FCA to represent and extract knowledge from a set of objects and attributes and it is even more clear the problem of dealing with databases of high dimensionality. Application in real problems often suffers from this common fact. In this work, an approach to meet the challenge mentioned above consists in applying Binary Decision Diagrams (BDDs) [4] to obtain a symbolic representation of a cross table (formal context) that allows a more efficient extraction of the set of all concepts. It will be shown that this approach is promising and that it can handle more efficiently with large contexts when compared with the conventional implementation of algorithms that handle standard tables.

Although BDD has already been used in the FCA to represent the concept lattice [5], this article focus in the representation of formal contexts in BDD to achieve faster concepts extraction. Different BDD libraries will be used to evaluate BDD, including the BuDDy [6] and CUDD [7] package. Both support several variable reordering methods and present a C++ interface that reference nodes automatically and dereference them accordingly by the garbage collector. In addition, CUDD has built-in Zero-Suppressed BDDs [8] capabilities that were also evaluated in this work.

This article is organized in five sections. In the second section, the main concepts of the FCA and BDD are reviewed. In the third section, examining the representation of formal contexts through BDD and the extraction of concepts from this implicit representation is discussed. In the fourth section, BDD is evaluated over several large formal contexts. In the last section, the conclusions and future works are pointed out.

2 Formal Context

2.1 Formal Concept Analysis

Formal Context. Formal contexts have the notation $K{:=}(G, M, I)$, where G is a set of objects (rows headers), M is a set of attributes (columns headers) and I is an incidence relation ($I \subseteq G \times M$). If an object $g \in G$ and an attribute $m \in M$ are in the relation I, it is represented by $(g, m) \in I$ or gIm and is read as "*the object g has the attribute m*".

Given a set of objects $A \subseteq G$ from a formal context $K{:=}(G, M, I)$, it could be asked which attributes from M are common to all those objects. Similarly, it could be asked, for a set $B \subseteq M$, which objects have the attributes from B. These questions define the derivation operators, which are formally defined as:

$$A' := \{m \in M \mid gIm \ \forall\ g \in A\} \ ; \ B' := \{g \in G \mid gIm \ \forall\ m \in B\} \tag{1}$$

A special case of derivate sets occurs when empty sets of objects or attribute are considered to be derivate:

$$A \subseteq G = \emptyset \Rightarrow A' := M \ ; \ B \subseteq M = \emptyset \Rightarrow B' := G \tag{2}$$

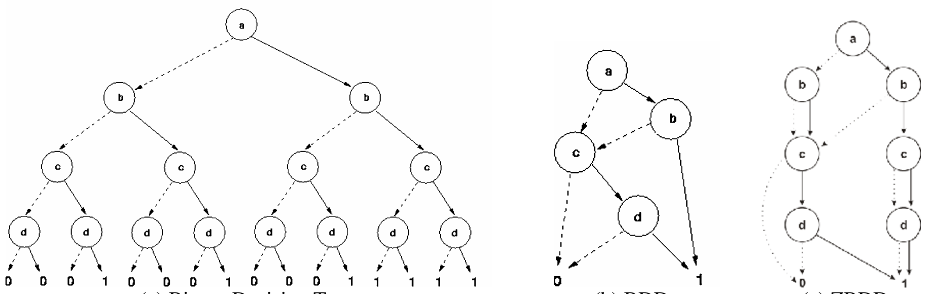
Formal Concept. Formal concepts are pairs (A, B), where $A \subseteq G$ (called extent) and $B \subseteq M$ (called intent). Each element of the extent (object) has all the elements of the intent (attributes) and, consequently, each element of the intent is an attribute of all objects of the extent. The set of all formal concepts in a formal context has the notation $\underline{\mathfrak{B}}(G, M, I)$. Since that a cross table representing a formal context is given, algorithms can be applied in order to determine its formal concepts and its lattice [9].

2.2 Binary Decision Diagrams

Binary decision diagrams are a canonical representation of boolean formulas [4]. The BDD is obtained from a binary decision tree by merging identical subtrees and eliminating nodes with identical left and right siblings. The resulting structure is a graph rather than a tree in which nodes are eliminated and substructures are shared.

Formally, a BDD is a directed acyclic graph with two types of vertex: non-terminal and terminal. Each non-terminal vertex is a distinct variable of the corresponding boolean formula. Also, each vertex has two outgoing arcs directed toward two children, corresponding to the case where variable is 0 *(left)* and 1 *(right)*. A BDD has two terminal vertices labeled by 0 and 1, representing the truth-value of the formula false and true, respectively. For every truth assignment to the boolean variables of the formula, there is a corresponding path from root to a terminal vertex.

Zero-Suppressed BDD (ZBDD) is a graph representation similar to a BDD. ZBDD also represents boolean formulas by elimination nodes and sharing subtrees. However, as opposed to BDD, it does not eliminate nodes whose two edges, left and right arcs, points to the same node. It presented another elimination rule in which all nodes that the right arc points to the zero terminal node are removed [8]. The BDD rule to merge identical subtrees is still present in ZBDD.



(a) Binary Decision Tree (b) BDD (c) ZBDD

Fig. 1. Graph representation for formula (a ∧ b) ∨ (c ∧ d)

Figure 1 illustrates a BDD and a ZBDD compared to a Binary Decision Tree for the boolean formula *(a ∧ b) ∨ (c ∧ d)*. Note in the ZBDD diagram that the nodes with two arcs pointing to the same node weren't removed like it would be with BDD. But nodes in ZBDD can still be removed when the new elimination rule is applied.

BDDs are an efficient way to represent boolean formulas. Often, they provide a much more concise representation compared to the traditional representations, such as conjunctive and disjunctive normal forms. BDDs are also a canonical representation for boolean formulas. This means that two boolean formulas are logically equivalent if and only if its BDDs are isomorphic. This property simplifies the execution of frequent operations, like checking the equivalence of two formulas.

However, BDD has drawbacks. The most significant is related to the order in which variables appear. Given a boolean formula, the size of the corresponding BDD is highly dependent on the ordering. It can grow from linear to exponential according to the number of variables of the formula. In addition, the problem of choosing a variable order that minimize the BDD size is NP-complete [4]. Despite the existence of heuristics to automatically order the variables, they are often ordered manually.

3 Formal Concepts Extraction Using BDD

When formal contexts are represented as BDDs, it is possible to implement derivation operators to work directly over this representation, thus allowing FCA algorithm independence. Unfortunately, the cost to identify the set of objects from a BDD concept is too expensive, thus invalidating this alternative. To overcome this problem, algorithms to extract concepts and/or to construct the concept lattice available in the literature must be adapted to handle this new form of representation.

To demonstrate the feasibility of BDD, the adapted algorithm was the Attribute Intersections [10] because of its inherent characteristics that allow a more effectively concepts extraction from contexts with more objects than attributes. This algorithm implementation in BDD was divided in three primary stages (Fig. 2). In the first stage, the construction of the context in BDD is made. The second stage is responsible to extract the set of all concepts from the BDD context. The final stage is responsible to identify the attributes and objects from the concepts represented in BDD. These stages separation avoids unnecessary high cost operations while obtaining the concepts.

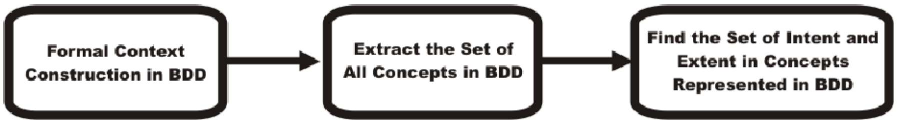

Formal Context Construction in BDD
Extract the Set of All Concepts in BDD
Find the Set of Intent and Extent in Concepts Represented in BDD

Fig. 2. Steps to implement the Attribute Intersection algorithm in BDD

3.1 Formal Context Construction in BDD

To create the BDD representation, a formal context must be converted into an equivalent logic boolean formula. Table 1 shows an example of a formal context and its possible representation through a logic function (Equation 3).

Table 1. Formal Context Example

	a1	a2	a3
o1	X		X
o2	X	X	
o3		X	

$$f(a_1, a_2, a_3) = a_1 \overline{a_2} a_3 + a_1 a_2 \overline{a_3} + \overline{a_1} a_2 \overline{a_3} \quad (3)$$

Note that each object is represented by a logic equation, according to the presence or not of its attributes. The function $f(a_1, a_2, a_3)$ results in a positive state (1) when an object is present on the context. This function returns the negative state (0) for objects not present in the context. Thus, any context can be represented by a logic function.

Algorithm 1 allows the construction of BDD based on the objects presented in the formal context. The internal functions *bdd_ithvar* and *bdd_nithvar* are specific to the library BuDDy [6] and are used to define the presence or not of an attribute in the BDD, respectively. The CUDD library has it own function wrappers to perform this operation, *Cudd_bddIthvar*. To obtain the respective negative variable, CUDD requires the use of *Cudd_Not* in conjunction with the *Cudd_bddIthvar*. Once the conjunction of attributes is made forming the objects (lines 7 and 9) then a disjunction of those objects is realized (line 12) to build the context.

Algorithm 1. BDD construction based on the context.

```
in:  List<Object> list
out: BDD context
 1: context = bddfalse
 2: while !list.empty( ) do
 3:   obj = list.removeFirstObject( );
 4:   BDD tmp = bddtrue
 5:   for i=0; i<obj.attributes; i++ do
 6:     if obj.hasAttribute(i) then
 7:       tmp &= bdd_ithvar(i)
 8:     else
 9:       tmp &= bdd_nithvar(i)
10:     endif
11:   done
12:   context |= tmp
13: done
```

Normally, ZBDDs are constructed by converting from a previously created BDD. However, to maintain coherence with the Algorithm 1, in this work, the ZDD was built object by object. CUDD maintains a different structure to operate ZBDD and positive variables reference may be achieved using the *Cudd_zddIthVar*. Since ZBDDs works with one's complement, it is not possible to use the standard *Cudd_Not* approach to obtain the negative part of a variable. But it can be obtained by the *Cudd_zddDiff* function, using the resulting difference from a 1-node constant, thus acquiring the complement.

In this work, references and dereferences of nodes were made manually when the CUDD library was used. In the other hand, the BuDDy C++ wrapper was used that allowed automatic references and dereferences of nodes. The garbage collector dereferences nodes automatically when memory resources are claimed.

It is important to emphasize that the main objective of this work is to show the feasibility of BDD to represent formal contexts, and from that representation extract the formal concepts. The feasibility is shown through the manipulation of large formal contexts. In most cases, the BDD representation of contexts often consumes several memory resources. However, it is not significant enough to invalidate this new representation in BDD. So the BDD can be used to extract concepts more efficiently than the algorithms that work directly in the tabular representation.

3.2 Extracting the Set of All Concepts in BDD

Algorithm 2 is the kernel of the Attribute Intersection algorithm, but slightly modified to work with BDD. This implementation in BDD takes advantage of two distinct moments when the derivation operator is used (Line 4) and the intersection between two concepts is made (Line 8). The derivation operator is easily implemented through the implicit *bdd_ithvar* operator, which obtains a BDD representation of all objects with an attribute. The intersection between two concepts is also implemented through an implicit BDD operation, *bdd_and* (&). Moreover, the concepts list was implemented as a *hashtable* to achieve a faster verification of concepts duplicity. The algorithm kernel is the same for the CUDD version with BDD and ZBDD.

Algorithm 2. BDD construction based on the context.

```
in:  BDD context
out: List<BDD> concepts
concepts = new List<BDD>
concepts.addConcept(context)
for i=0; i<attributes; i++ do
  BDD tmp1 = context & bdd_ithvar(i)
  size = concepts.size()
  for j=0; j<size; j++ do
    BDD tmp2 = concepts.getConcept(j)
    BDD intersection = tmp1 & tmp2
    if !concepts.exist(intersection) then
      concepts.add(intersection)
    endif
  done
done
```

Unfortunately, storing all the concepts as BDD in the list reflects a very expressive memory consumption. The algorithm was slightly modified to save the concept intent (B_i) rather than the concept (A_i, B_i) in BDD. From the intent set (B_i), one can rebuild the concept in BDD, thereby maintaining the essence of the proposed Algorithm 2.

3.3 Finding the Set of Intent and Extent in Concepts Represented in BDD

This section shows how to obtain the extent and intent of the concepts represented in BDD. Algorithm 3 is used to check if all objects represented by the BDD share an attribute in common. Algorithm 4 is used to verify whenever an object is present in the BDD concept.

For the extraction of all objects (extent) of the concept, Algorithm 4 can be used to verify if each object that exists in the formal context is present in the concept. The same can be applied to the set of attributes (intent), through Algorithm 3, covering all formal context attributes checking whether or not they are present in the concept. Also, in Algorithm 4, the BuDDy *bdd_varlevel* operation has no correspondence in the CUDD library, but can be obtained by their node index value.

Algorithm 3. Verify the presence of an attribute in a concept represented in BDD.

```
in:  BDD concept, attr
out: presence
BDD tmp = concept & bdd_ithvar(attr)
if tmp == concept then
  present = true
else
  present = false
endif
```

Algorithm 4. Verify the presence of an object in a concept represented in BDD.

```
in:  BDD concept, objc
out: presence
 1: BDD tmp = concept
 2: i = 0
 3: while i<objc.attributes and
 4:       tmp != {bddtrue, bddfalse} do
 5:   if bdd_varlevel(tmp) == i then
 6:     if obj.hasAttribute(i) then
 7:       tmp = bdd_high(tmp)
 8:     else
 9:       tmp = bdd_low(tmp)
10:     endif
11:   endif
12:   i++
13: done
14: presence =(tmp == bddtrue)
```

Although Algorithm 4 is used to identify if an object exists in a BDD concept, it was not used in the Attribute Intersection implementation. As mentioned, only the concepts intents are store in the list. With the intents it is not necessary to rebuild the BDD concept to verify objects presence. It can be done directly by checking if every object has the attributes of each of the intent sets on the list.

4 Feasibility Analysis of BDD to Extract Concepts

One of the requirements to assess the representativeness of BDD to extract concepts was to compare its performance under the same conditions as its tabular version. For this reason, it was decided to implement a unique algorithm for the situations: contexts represented in BDD (with BuDDy and CUDD), ZBDD (with CUDD) and by a table. Also, several optimizations on the table version were made to meet this purpose. Moreover, SCGaz (available at http://www.inf.pucminas.br/projetos/licap) was the tool responsible to build all contexts used in this work. This tool allows the construction of semi-clarified contexts, avoiding some types of attributes and objects redundancy, like repeated objects and also empty and full attributes and objects.

Figure 3 shows the behavior of the Attribute Intersections algorithm for the BDD with BuDDy and CUDD, ZBDD and tabular version for contexts with fewer attributes (20 to 100) and many objects (10,000 to 60,000). To ensure that the BDD would not be extremely compact, the used density for all contexts was the minimum plus 10% of it. Moreover, lower density values result into smaller amounts of concepts, thus making the simulations consumes less time to execute. All software were written in C++ and executed on a Pentium Dual Core 2.66GHz with 2Gb of RAM running Linux Slackware 12.0.

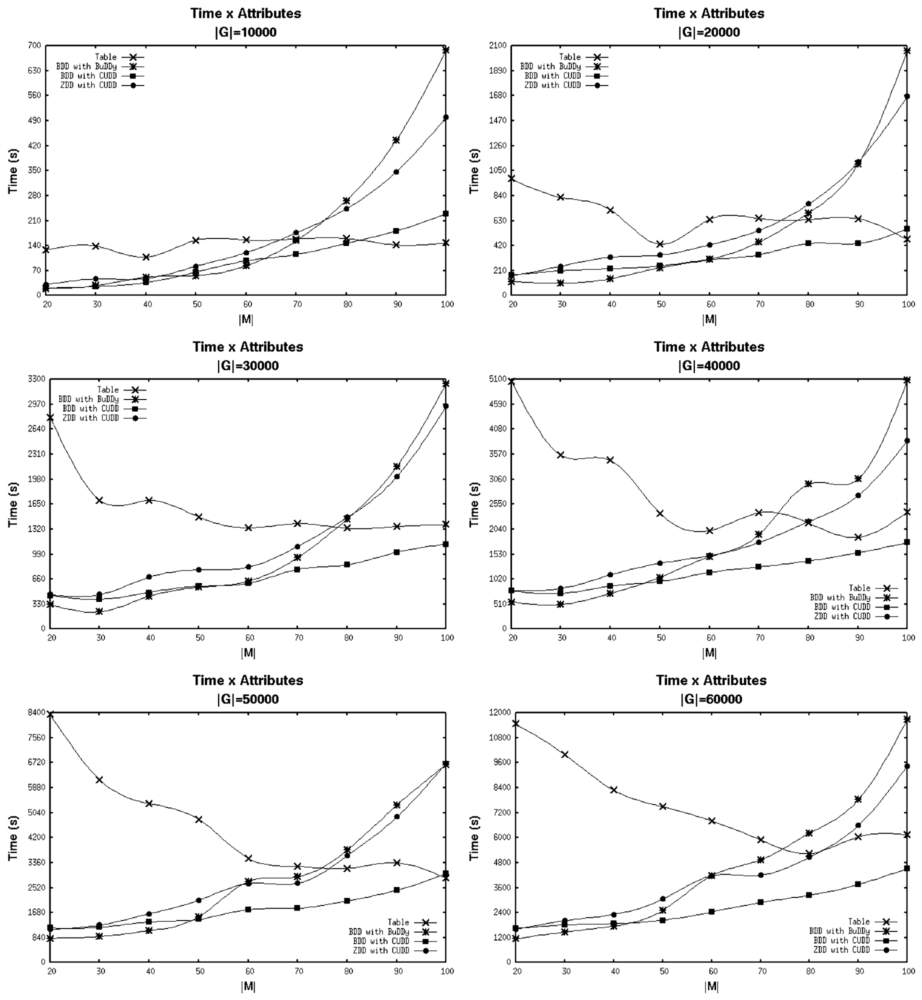


Fig. 3. Evaluation of Attribute Intersections implemented as a table, BDD and ZBDD

The tabular version has presented an irregular decreasing behavior because of the density, since with few attributes higher will be the density for these considered simulations. In addition to that, how the incidences are spread into the context can explain its irregular behavior. The BDDs and ZBDD had exponential behavior, even the CUDD version that may appear linear. Increase the attributes may allow to observe its exponential behavior. With more attributes, more nodes will be required to construct the BDD. Therefore, less efficient will be the operations in this representation. Also, as can be seen by simulations of 20 and 30 attributes, while the tabular version had worst time performance, the BDD maintained a very low execution time, despite of the higher density. So, the context BDD size is extremely relevant to the extraction of all concepts.

Table 2. Execution time for many-valued context with |M|=70, |G|=60,000 and |g'|=7

	Construction of the Context (s)	Concepts Extraction (s)	Intent and Extent Identification (s)	Total (s)	Total
Table	-	-	-	81059	0d 22:30:57
BuDDy BDD	51	10151	9946	20148	0d 05:35:48
CUDD BDD	192	14075	10107	24374	0d 06:46:14
CUDD ZBDD	306	10272	9703	20281	0d 05:38:01

Table 3. Execution time for many-valued context with |M|=70, |G|=120,000 and |g'|=7

	Construction of the Context (s)	Concepts Extraction (s)	Intent and Extent Identification (s)	Total (s)	Total
Table	-	-	-	251283	2d 21:48:03
BuDDy BDD	128	18345	33289	51762	0d 14:22:42
CUDD BDD	569	36841	33043	70453	0d 19:34:13
CUDD ZBDD	629	22107	34844	57580	0d 15:59:40

The BDD with CUDD package had better performance over the table in all simulations, making it strongly reliable. It would be necessary more simulations with higher attributes to verify if there is a threshold in which the table begins to be more worthed. BDD with BuDDy and ZBDD had better performance over table when the number of concepts is not superior to 70. As the amount of objects increases, greatest has become the difference between the execution times of BDD implementations compared to the table. Thus, the implicit representation of concepts in BDD becomes an alternative to a more efficient extraction of concepts in these conditions.

Considering now a threshold of 70 attributes, another simulation scenario was created. A many-valued context was simulated with 7 attributes, a fixed number of 10 attribute-values per attribute and in two situations with objects, 60,000 and 120,000. This type of context has 7 attributes per object (|g'|=7). Table 2 and 3 presents the spent time consumed by these situations.

For this type of context, BDD and ZBDD had outstanding improvement over the tabular version. As opposed to the previous simulations, the BDD with CUDD presented the least performance. In spare context, ZBDD was proven faster, but not enough to beat BuDDy performance. Also, CUDD required more time to build the context, possibly affecting the final BDD size. BDD with BuDDy and ZBDD had similar execution times for contexts with 60,000 objects, but taking the ICFCA'06 challenge with 120,000 objects, the difference was quite significant, over 1 hour and a half. Compared to the table version, the difference between the BuDDy version was almost three days. Applicability to process larger contexts could be achieved with the use of a distributed version of an algorithm implemented in BDD.

Note that the required time to identify the set of extents from the concepts represented in BDD was very significant, as seen by Table 2 and 3. This happens because of the used algorithm quadratic complexity relative to the number of objects. If more efficient algorithms were used, lower computational times may be achieved to process contexts. Instead of a brute force strategy to check objects presence in a concept, another strategy could be visiting BDD nodes in order to identify the objects.

5 Conclusions

The present work is related to a challenge raised at the ICFCA'06 conference, which refers to the manipulation of large formal contexts. Through the use of an implicit representation of formal context in BDD, it has been demonstrated that this new representation became computationally feasible for handling large contexts, when compared to the conventional manipulation of a table. Although the BDD allows the manipulation of contexts with a large number of objects, it is restricted to contexts with few attributes. Thus, if the context meets this feature, a significant efficiency can be achieved with the application of this new alternative.

As future work, more robust FCA algorithms could be adapted to use BDD or ZBDD. It's essential to verify the feasibility of BDD application in a fast algorithm, like CHARM [11]. Even faster concepts extraction may be achieved with others algorithms applied with BDD.

References

1. Li, Y., Liu, Z.T., Shen, X.J., Wu, Q., Qiang, Y.: Theoretical research on the distributed construction of concept lattices. In: International Conference on Machine Learning and Cybernetics, vol. 1, pp. 474–479 (2003)
2. Liu, Z., Li, L., Zhang, Q.: Research on a union algorithm of multiple concept lattices. In: Wang, G., Liu, Q., Yao, Y., Skowron, A. (eds.) RSFDGrC 2003. LNCS (LNAI), vol. 2639, pp. 533–540. Springer, Heidelberg (2003)
3. Lévy, G., Baklouti, F.: A distributed version of the ganter algorithm for general galois lattices (2005)
4. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
5. Yevtushenko, S.: Computing and visualizing concept lattices. PhD thesis, TU Darmstadt, Fachbereich Informatik (2004)
6. Lind-Nielsen, J.: Buddy: A binary decision diagram. Technical report, Department of Information Technology, Technical University of Denmark, Lyngby, Denmark (1999), `http://www.itu.dk/research/buddy`
7. Somenzi, F.: CUDD: CU decision diagram package release (1998)
8. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: DAC 1993: Proceedings of the 30th International Conference on Design Automation, pp. 272–277. ACM, New York (1993)
9. Grätzer, G.: General Lattice Theory. Birkhäuser, Basel (1978)
10. Carpineto, C., Romano, G.: Concept Data Analysis: Theory and Applications. John Wiley & Sons, Indianapolis (2004)
11. Zaki, M., Hiao, C.: ChARM: An efficient algorithm for closed association rule mining. Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Inst., Troy, NY, USA (October 1999)

Comparing Genetic Algorithms and Newton-Like Methods for the Solution of the History Matching Problem

Elisa Portes dos Santos[1,*],Carolina Ribeiro Xavier[1,*],
Paulo Goldfeld[2], Flavio Dickstein[2],
and Rodrigo Weber dos Santos[1]

[1] Dept. of Computer Science, Federal University of Juiz de Fora
Juiz de Fora, MG, Brazil
[2] Dept. of Applied Mathematics, Federal University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
* First Authors in Alphabetical Order
rodrigo.weber@ufjf.edu.br

Abstract. In this work we presents a comparison of different optimization methods for the automatic history matching problem of reservoir simulation. The history matching process is an inverse problem that searches a set of parameters that minimizes the difference between the model performance and the historical performance of the field. This model validation process is essential and gives credibility to the predictions of the reservoir model. Derivative-based methods are compared to a free-derivative algorithm. In particular, we compare the Quasi-Newton method, non-linear Conjugate-Gradient, Steepest-Descent and a Genetic Algorithm implementation. Several tests are performed and the preliminary results are presented and discussed.

Keywords: Reservoir simulation, History Matching, Optimization.

1 Introduction

Reservoir simulation is a powerful tool that has been extensively used in reservoir engineering. It combines physics, mathematics, reservoir engineering and computer programming. One of the main goals of the models deals with the ability to predict the behavior of a reservoir. Unfortunately, the computational models depend on many parameters and features of the reservoir and the prediction's performance of a model depends on good estimations of some physical properties, such as the permeability distribution of the reservoir. Several difficulties arise during the validation of a model, since most of the oil reservoirs are inconveniently buried beneath thousands of feet of overburden. Direct observations of the reservoir are available only at well locations that are often hundreds of meters [1].

An alternative for model validation is the estimation of the relevant properties by History Matching[2]. The History Matching process is an inverse problem that utilizes reservoir simulation to find a set of parameters that minimizes the difference between the model performance and the historical performance of the field. This process can be made manually or automatically.

Traditional Newton-like methods have been used before[3]. In addition, free-derivative methods based on Genetic Algorithms were proposed in[4]. This paper presents a comparison of different optimization methods used to perform the automatic history matching in a 2D flow model. In particular we compare the Quasi-Newton method, Conjugate-Gradient, Steepest-Descent and a Genetic Algorithm implementation.

The paper is organized as follows: Section 2 introduces the direct problem formulation and implementation. Section 3 introduces the inverse problem theory and the methods implemented. Section 4 presents the methods and the computer platform used for the tests. Section 5 and 6 present the results and conclusion of this work, respectively.

2 Forward Problem

2.1 Theory

The problem treated in this paper is a two dimensional two-phase (water/oil) incompressible and immiscible porous media flow in a gravity-free environment [12]. The system of partial differential equations which governs this flow is derived from the *law of mass conservation* and the *Darcy Law*. The law of mass conservation for both phases is written as $\phi\partial_t(\rho_\alpha s_\alpha) + \nabla.(\rho_\alpha v_\alpha) = Q_\alpha$, where $\alpha = w$ denotes the water phase, $\alpha = o$ denotes the oil phase, ϕ is the porosity of the porous medium, and ρ_α, s_α, v_α and Q_α are, respectively, the density, saturation, volumetric velocity and flow rate in wells of the α-phase. The volumetric velocity (v_α) is given by the Darcy law: $v_\alpha = \frac{Kk_{r\alpha}(s_\alpha)}{\mu_\alpha}\nabla p_\alpha$, where K is the effective permeability of the porous medium, $k_{r\alpha}$ is the relative permeability of α-phase, which is a function that depends on saturation, and μ_α and p_α are, respectively, viscosity and pressure of the α-phase. In this work we consider that the capillary pressure is null, that is, $p_w = p_o$. So, from now on we will refer to pressure simply as p. We also have that $s_w + s_o = 1$. We introduce the phase mobility and transmissibility functions, respectively: $\lambda_\alpha(s) = \frac{k_{r\alpha}(s)}{\mu_\alpha}$, $T_\alpha(s) = K\lambda_\alpha$, where $s = s_w$ from now on. The volumetric velocity can then be written as $v_\alpha = -T_\alpha\nabla p$. We assume that the phases density and viscosity are constant and get

$$\begin{cases} \phi\rho_w\partial_t s_w + \rho_w\nabla v_w = Q_w \\ \phi\rho_o\partial_t s_o + \rho_o\nabla v_w = Q_o \ . \end{cases} \tag{1}$$

Now we can divide the equations in 1 by ρ_α and sum both and get

$$\begin{cases} \phi\partial_t s + \nabla v_w = q_w \\ \nabla v_t = q_t \ . \end{cases} \tag{2}$$

where $q_\alpha = \frac{Q_\alpha}{\rho_\alpha}$ is the flow rate density of α-phase, $q_t = q_w + q_o$ and $v_t = v_w + v_o$. Defining total mobility as $\lambda_t = \lambda_w + \lambda_o$ we introduce the fractional flow functions as $f(s) = \frac{T_w}{T_t} = \frac{\lambda_w}{\lambda_t}$. System 1 is then rewritten as

$$\begin{cases} \phi \partial_t s - \nabla(f(s)T_t(s)\nabla p) = q_w \\ -\nabla(T_t(s)\nabla p) = q_t \ . \end{cases} \tag{3}$$

To complete the model the boundary conditions must be specified. In this paper we consider no flow boundary condition, $v_\alpha . \nu = 0, x \in \partial\Omega$, where ν is the outer unit normal to the boundary $\partial\Omega$ of the domain Ω. Finally we define the initial condition given by $s(x, 0) = s_0(x), x \in \Omega$.

The forwad problem treated on this paper is the system of partial differential equations given by 3 with the boundary and initial conditions given above.

2.2 Implementation

The differential equations described in Sect. 2.1 are nonlinear and coupled. In this work the method used to solve these equations is the so called IMPES. Our implementation of the IMPES methods adopts an adaptive time step scheme. The basic idea of the IMPES method is to separate the computation of pressure from that of saturation. The coupled system is split into a pressure equation and a saturation equation, and the pressure and saturation equations are solved using implicit and explicit time approximation approaches, respectively. Decoupling the system 3 we get an elliptic equation for pressure given by (4) and a nonlinear hyperbolic equation for saturation, given by (5).

$$-\nabla(T_t(s)\nabla p) = q_t \ . \tag{4}$$

$$\phi \partial_t s - \nabla(f(s)T_t(s)\nabla p) = q_w \ . \tag{5}$$

For the pressure computation, the saturation s in (4) is supposed to be known and 4 is solved implicitly for p. In this work, the finite volume method was used for spatial discretization[5]. As mentioned before, the saturation equation given by 5 is solved explicitly. The IMPES method goes as follows: given s^0; for $n = 0, 1, ...$ we use (4) and s^n to evaluate p^n; next we use (5), s^n and p^n to evaluate s^{n+1}. To guaranty the stability of this equation the time step Δt must be sufficiently small which is an expensive requirement. To minimize this problem, we actually used an Improved IMPES method [6]. This method uses the fact that pressure changes less rapidly in time than saturation. Knowing this, it is appropriate to take a much larger time step for the pressure than for the saturation. Using the Improved IMPES method we have two different time steps: Δt^n for pressure and $\Delta t^{n,l}$ for saturation. Pressure p^n corresponds to instant $t^n = \sum_{1<=i<=n} \Delta t^i$ and saturation $s^{n,l}$ corresponds to instant $t^{n,l} = t^n + \sum_{1<=j<=l} \Delta t^{n,j}$. We deduced the CFL conditions given by the next two equations. One for cells that have injector wells $\Delta t^{n,l} \leq \frac{\phi(1-s_{o,res}-s^{n,l}_{i,j})}{\beta_1 q_w^{n,l}(1-f(s^{n,l}_{i,j}))}$, where

$\beta_1 > 1$ and another to the other cells, given by $maxf'(s_m)\sum_m \frac{\Delta t^{n,l}}{\phi \Delta_m}|v_m^n| \leq \beta_2$, where $0 < \beta_2 < 1$ and m corresponds to interfaces where the flow enters the block. To control the pressure time step Δt^n we calculate the pressure variation percentage $VP^n = \frac{||p^{n+1} - p^n||}{||p^n||}$. If VP^n is greater than a given VP_{max} pressure time is reduced and if it is less then a given VP_{min}, it is increased.

3 Inverse Problem

3.1 Theory

In this work, the inverse problem proposed aims to estimate the absolute permeability field of a reservoir by history-matching its production data. We denote by K the vector of permeability to be determined and by O the vector of production observations and define as $u = (s, p)$ the vector of the forward problem unknowns (saturation and pressure). We have that u depends on the permeability $u = u(K)$ and O depends on both, permeability and u, $O(K) = O(K, u(K))$. If $\bar{O}$ is the vector with the real observations we can search K that minimizes the least square formulation

$$f(K) = \|O(K) - \overline{O}\|^2. \tag{6}$$

Note however that this is a contrained minimization problem since permeability is a strictly positive property. In this work, we transformed this problem in an unconstrained minimization problem via the change of variable $m_i = ln(K_i)$. From now on, the parameters to be estimated are those of the vector m. There are several ways for solving this optimization problem. In this work we compare two different approaches to optimize the proposed problem: Newton-like methods and Genetic Algorithm.

3.2 Newton-Like Methods

Newton's iterative method for minimizing $f(x)$ is $x_{k+1} = x_k + \alpha_k H_k^{-1} \nabla f^T$, where H_k is the Hessian matrix of f in iteration k. However the computation of H is almost allways unaffordable. The idea underlying Newton-like methods is to use an approximation to the inverse of the Hessian in place of the true inverse that is required on Newton's method [7]. In this work we use the *steepest descent*, the *conjugate gradient* and a *Quasi-Newton* methods. In steepest descent method, the inverse of H is taken as the identity matrix I. The Quasi-Newton Method iteratively builds up an approximation to the Hessian by keeping track of gradient differences along each step taken by the algorithm. The nonlinear conjugate gradient method is mainly an extension of the linear conjugate gradient method but it may be also consider a specialization of limited-memory Quasi-Newton methods.

3.3 Genetic Algorithms

Genetic Algorithm (GA) is a numerical optimization algorithm inspired in natural selection and natural genetics created by Holland in the 60th decade [8]. It is an alternative optimization technique since it is stochastic and does not need derivatives. In each generation, which corresponds to an iteration of the algorithm, GA has not only one possible solution. Instead, it keeps a population of individuals each representing a potential solution to the problem to be solved. This population evolves through generations using genetic mechanisms in search for an optimal individual. In order to use GA it is important to define some characteristics such as the representation of an individual, a fitness function and the genetic operators to be used. In this work we used the real-code representation which is commonly used in problems with continuous variables. In this representation the individual is a set of real values that represents a feasible solution. Every individual in the population is assigned, by means of a fitness function, a measure of its goodness with respect to the problem under consideration [8]. The fitness function used in this work is given by (6). The most important genetic operators applied in GA are selection, crossover, mutation and elitism. Selection is used to pick individuals to generate offsprings. The approach used in this work is the rank-based selection. Crossover is the operator that combines two individuals to generate a third one. In this work we used the *Blended-crossover* approach. Mutation is an operator applied in one individual. It introduces diversity to the population and prevents the algorithm of being stuck in local minimums. The GA begins with an initial population that may or not be generated randomly. To evolve to a next generation all individuals of the population must be evaluated using the fitness function. After this, selection is applied to choose which individuals will generate offsprings through crossover or/and mutation. This process is repeated until a good individual is found.

4 Methods

4.1 Implementation Details and Computer Platform

The numerical solution of the forward problem was implemented in C++. To solve the linear systems associated to the discretization of the Partial Differential Equations the PETSc library was used [9]. The GSL library [10] was used for the optimization with the Newton-like methods. In this work we used the *steepest descent*, *nonlinear conjugate gradient* and *Quasi-Newton* methods. These methods need the objective function (6) and its gradient ∇f to be evaluated for each solution candidate m. The objective function is calculated via the solution of the forward problem. The calculation of the gradient ∇f is obtained via finite difference and it involves performing $n + 1$ solutions of the forward problem, where n is the dimension of the permeability vector to be estimated. The calculation of the gradient ∇f was implemented in parallel using the MPI library[11] with each component of the gradient vector calculated by a different process. The GA was implemented in C++. The implementation also exploits parallelism via the MPI library and a master-slave decomposition strategy. The master process implements all the GA operations and requests the slave processes to perform the fitness evaluation of the solution candidates, or individuals, of the current population. Again, each fitness calculation given by (6) involves the solution of the forward problem. The algorithms were executed in a small cluster composed of 8 Intel Xeon (2GHz) processing cores connected by a 1000MBits Ethernet switch.

4.2 Numerical Experiments

The reservoir simulation we consider in this work is the classical five-spot configuration with 4 injection wells in the corners of the reservoir and one production well in its center (see Figure 1). The reservoir is a square of sizes equal to $200m$. Each injection well injects a total of $100m^3$ per day. The reservoir's history is given by the oil production of the center well during 350 days of simulation. The other parameters of the model are: porosity (0.2), relative permeability, given by the Corey curve, irreducible water saturation ($s_{iw} = 0.2$) and residual oil saturation ($s_{ro} = 0.2$). The spatial discretization used was.$\Delta x = \Delta y = 7.4m$. In this work, three different synthetic histories were generated from different reservoirs that differ only on the number of rectangular regions with different permeability values: a reservoir model with two different permeability blocks mapped as a 1x2 mesh (half by half), $K = (47.82, 142.5)$ (natural ordering); a model with an uniform 2x2 permeability mesh, $K = (39.12, 67.99, 52.85, 267.68)$; and an uniform 3x3 permeability mesh with $K = (43.22, 38.21, 55.76, 56.39, 95.61, 148.45, 36.84, 135.84, 261.57)$ These three synthetic histories are the targets of three different history-matching problems. Each one of these inverse problem was solved by the Newton-like methods and the GA, and each of the optimization methods were executed 10 times with different initial guesses. The population of the GA

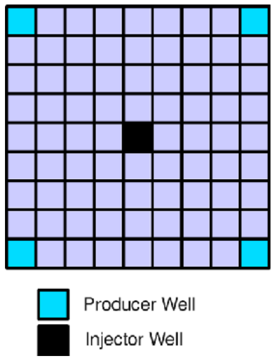


Fig. 1. 5-spot

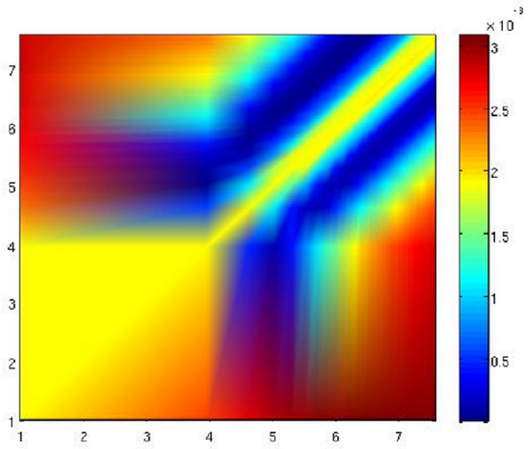


Fig. 2. Objective Function

has a size of 100 and the mutation rate was 0.025. Convergence is achieved when a solution candidate m satisfies $f(m) < 10^{-6}$. All initial guesses were randomly generated but satisfy $f(m) > 10^{-3}$. Therefore, after convergence the objective function is decreased by at least 3 orders of magnitude. In the case of the Newton-like methods there is also a stop criterion to handle local minima: $\nabla f(m) < 10^{-8}$. In addition, for all methods we implemented a stagnation criterion: the method stops if after 3 consecutive iterations $f(m)$ is decreased by less than 10^{-6}.

5 Results

Figure 2 shows the objective function for the problem that has two parameters or permeability values to be estimated. We note that there are infinitely many global minima and that the shape of the function is symmetric with respect to the line $m = \alpha(1,1)$. The reason we have many global minima lies in the following observation. Imagine two simulations where in the second one the permeability is given by $K_2 = \beta K_1$, with K_1 the permeability of the first simulation. Let us further identify this simulations with s_1, p_1, K_1 and s_2, p_2, K_2. It follows that from (4) we have $\nabla(f(s_2)T_{t2}(s_2)\nabla p_2) = \nabla(f(s_2)K_2\lambda_t(s_2)\nabla p_2) = \beta\nabla(f(s_2)K_1\lambda_t(s_2)\nabla p_2) = \nabla(f(s_1)K_1\lambda_t(s_1)\nabla p_1)$. Using this in (5) we observe that for both simulations the saturation equation becomes exactly the same. Thus, the production history will be the same for these simulation with different permeability maps. Now, the fact that the shape of the objective function is symmetric is a particular feature of the simulation we perform, the classical five-spot. Since water injection is the same in all injection wells, 180^o rotations of the permeability distribution map do not alter the production of the centered well.

For the simulations with the two parameters to be estimated we have performed 4 different set of tests. Set 1, 2, 3 and 4 have initial guesses (initial population for the GA) randomly chosen but that satisfy $f(m) \geq 2.5\,10^{-3}$, $f(m) \geq 2.0\,10^{-3}$, $f(m) \geq 1.5\,10^{-3}$ and $f(m) \geq 1.0\,10^{-3}$, respectively. For the Newton-like methods, the initial guess was taken as the permeability pair with the smallest objective function value in the corresponding GA initial population. In each test the algorithms were executed 10 times. Table 1 presents the results in terms of successfull convergence of the methods. The GA is the most robust method. It has always converged to a global minimum. The performance of the Newton-like methods clearly depended on the proximity to a global minima. The performance improves if the initial guess is close to the minima. The Steepest-Descent (SD) achieved the worst result.

Table 2 presents some statistics of the results for set # 3. The general behavior of the methods were similar for the other test sets. The results obtained by the GA in terms of error (fitness) average and standard deviation (std) are at least an order of magnitude smaller than those obtained by the CG (Conjugate Gradient) and QN (Quasi-Newton) methods and two orders of magnitude better than those of the SD method (Steepest Descent). However, the computational cost of the

Table 1. Successfull convergence with 2 parameters

Initial guess (m)	GA	CG	QN	SD
$f(m) \geq 2.5\,10^{-3}$	10	3	3	2
$f(m) \geq 2.0\,10^{-3}$	10	4	4	4
$f(m) \geq 1.5\,10^{-3}$	10	6	6	3
$f(m) \geq 1.0\,10^{-3}$	10	7	7	4

Table 2. 2 parameters statistics- initial guess m - $f(m) \geq 1.5\,10^{-3}$

	Best fit	Mean	Std	# of eval. (fastest)	Total # eval.
GA	1.587007e-09	1.4882e-07	2.0064e-07	208	3236
CG	2.868986e-16	6.6401e-05	9.9601e-05	32	486
QN	3.400530e-17	6.6401e-05	9.9601e-05	32	462
SD	5.446458e-07	1.3550e-04	1.0114e-04	25	700

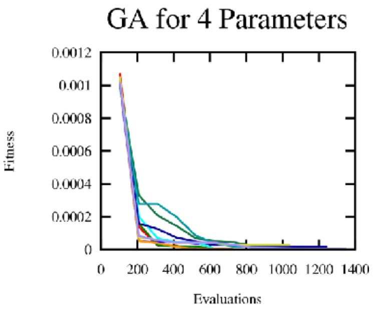
GA in terms of total number of objective function evaluations is around 8 (6) times higher than that of the CG and QN methods (SD).

For the tests with 4 and 9 parameters we chose to compare the GA against the CG method, as this method and the QN method achieved very similar performance in the tests with 2 parameters. The two algorithms were executed 10 times with different initial guesses for the inverse problem with 4 parameters. The GA successfully converged 6 times whereas the CG converged only once to the desired error tolerance of 10^{-6}. Table 3 presents some statistics of the results. The results obtained by the GA in terms of error average and standard deviation (std) are near two order of magnitudes smaller than those obtained by the CG. However, the computational cost of the GA is around 6 times higher than that of the CG. The fastest execution of the GA needed 10 times more evaluations of the objective function than the CG method. Figure 4 presents the evolution of the executions of the GA and CG methods. The GA is quite robust in terms of convergence and the executions perform similarly whereas the evolution of a CG execution highly depends on the initial guess.

Table 4 presents some statistics of the results obtained by the methods for the History-matching problem with 9 parameters. The two algorithms were executed 10 times with different initial guesses. The CG successfully converged twice whereas the GA converged only once. Nevertheless, the results obtained by the GA in terms of error average and standard deviation are smaller than those obtained by the CG. However, the computational cost of the GA is around

Table 3. Statistics of the tests with 4 parameters

	Best fit	Mean	Std	# of eval. (fastest)	Total # eval.
GA	1.2590e-07	4.7753e-06	5.4781e-06	624	18489
CG	3.000341e-07	1.7023e-04	2.7912e-04	62	2989



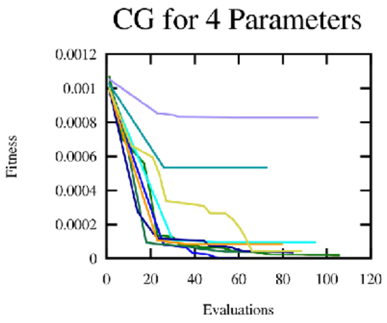
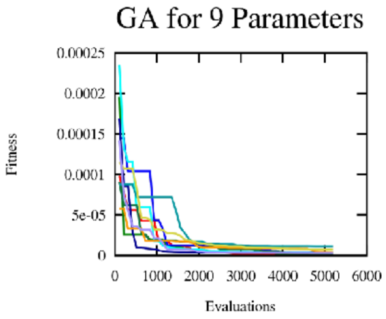


Fig. 3. GA performance for 4 parameters **Fig. 4.** CG performance for 4 parameters

Table 4. Statistics of the tests with 9 parameters

	Best fit	Mean	Std	# of eval. (fastest)	Total # eval.
GA	9.479320e-07	3.8718e-06	3.0535e-06	4160	50960
CG	9.197778e-07	6.1349e-05	8.2611e-05	252	3940



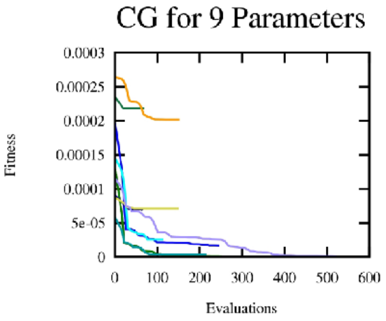


Fig. 5. GA performance for 9 parameters **Fig. 6.** CG performance for 9 parameters

13 times higher than that of the CG. Figure 6 presents the evolution of the executions of the GA and CG methods. Again, the evolution of the GA executions were very similar whereas the evolution of a CG execution highly depends on the initial guess.

6 Conclusion

This work presents a comparison of different optimization methods for the automatic history matching problem of reservoir simulation. The computational model implemented was based on the formulation of a two dimensional two-phase (water/oil) incompressible and immiscible flow in a gravity-free porous media. The reservoir simulation considered was the classical 5-spot configuration. The inverse problem proposed aims to estimate the absolute permeability distribution of the reservoir by history-matching its production data. Derivative-based methods were compared to a free-derivative algorithm. In particular, we compared the Quasi-Newton method (QN), non-linear Conjugate-Gradient (CG), Steepest-Descent (SD) and a Genetic Algorithm (GA) implementation. The GA presented the most consistent results in terms of accuracy and convergence but it was computationally very expensive. On the other side, the performances of CG and QN methods were very dependent on the initial guesses. Their results were less consistent, but the methods demanded few evaluations of the objective function and therefore few executions of the reservoir simulator. In conclusion, the preliminary results suggests a tradeoff in terms of robustness, towards the Genetic Algorithm, and speed, which favors the Newton-like methods.

Acknowledgment. The authors thank PETROBRAS for the financial support.

References

1. Oliver, D.S., Albert, C.R., Liu, N.: Inverse theory for Petroleum Reservoir Characterization and History Matching. Cambridge University Press, Cambridge (2008)
2. Watson, A.T., Wade, J.G., Ewing, R.E.: Parameter and system identification for fluid flow in underground reservoirs. In: Conference on Inverse Problems and Optimal Design in Industry, Philadelphia (1994)
3. Brun, B., Gosselin, O., Barker, J.W.: Use of Prior Information in Gradient-Based History-Matching. In: SPE Reservoir Simulation Symposium, pp. 13–23 (2001)
4. Soleng, H.: Oil reservoir production forecasting with uncertainty estimation using genetic algorithms. In: Congress on Evolutionary Computation, pp. 1217–1223 (1999)
5. Versteeg, H., Malalasekra, W.: An Introduction to Computational Fluid Dynamics: The Finite Volume Method, 2nd edn. Prentice Hall, Harlow (2007)
6. Chen, Z., Huan, G., Li, B.: An improved IMPES method for two-phase flow in porous media. Transport in Porous Media 32, 261–276 (2004)
7. Luenberg, D.G.: Introduction to Linear and Nonlinear Programming. Addison-Wesley, Reading (1973)
8. Coley, D.A.: An Introduction to Genetic Algorithms for Scientists and Engineers. World Scientific Publishing Company, River Egde (1997)
9. Balay, S., Buschelman, K., Gropp, W.D., et al.: PETSc users manual. Technical Report ANL-95/11, Argonne National Laboratory (2002)
10. Galassi, M., Davies, J., Theiler, J., et al.: GNU Scientific Library Reference Manual. Network Theory, Bristol (2006)
11. Message Passing Interface Forum: MPI, a message-passing interface standard. Oregon Graduate Institute School of Science & Engineering (1994)
12. Chen, Z.: Reservoir Simulation: Mathematical Techniques in Oil Recovery. Society for Industrial and Applied Mathematics (2007)

Complex System Simulations with QosCosGrid

Krzystof Kurowski[4,5], Walter de Back[2], Werner Dubitzky[3], Laszlo Gulyás[1,2], George Kampis[2], Mariusz Mamonski[4], Gabor Szemes[1,2], and Martin Swain[3]

[1] AITIA International Inc., Budapest, Hungary
[2] Collegium Budapest – Institute for Advanced Study, Budapest, Hungary
[3] University of Ulster, Coleraine, United Kingdom
[4] Poznan Supercomputing and Networking Center, Poznan, Poland
[5] University of Queensland – Institute for Molecular Bioscience, Australia
krzysztof.kurowski@man.poznan.pl, wdeback@colbud.hu, w.dubitzky@ulster.ac.uk, lgulyas@colbud.hu, gkampis@colbud.hu, mamonski@man.poznan.pl, gszemes@colbud.hu, mt.swain@ulster.ac.uk

Abstract. The aim of the QosCosGrid project is to bring supercomputer-like performance and structure to cross-cluster computations. To support parallel complex systems simulations, QosCosGrid provides six reusable templates that may be instantiated with simulation-specific code to help with developing parallel applications using the ProActive Java library. The templates include static and dynamic graphs, cellular automata and mobile agents. In this work, we show that little performance is lost when a ProActive cellular automata simulation is executed across two distant administrative domains. We describe the middleware developed in the QosCosGrid project, which provides advance reservation and resource co-allocation functionality as well as support for parallel applications based on OpenMPI (for C/C++ and Fortran) or ProActive for Java. In particular, we describe how we modified ProActive Java to enable inter-cluster communication through firewalls. The bulk of the QosCosGrid software is available in open source from the QosCosGrid project website: www.qoscosgrid.org.

Keywords: Grid computing, complex system, parallel applications, ProActive, Java, advance reservation, co-allocation, modeling and simulation.

1 Introduction

Complex systems are defined as systems with many interdependent parts which give rise to non-linear and emergent properties determining the high-level functioning and behavior of such systems [1]. Due to the interdependence of their constituent elements and other characteristics of complex systems, it is difficult to predict system behavior based on the ‘sum of their parts’ alone. Examples of complex systems include bee hives, bees themselves, human economies and societies, nervous systems, molecular interactions, cells and living things, ecosystems, as well as modern energy or telecommunication infrastructures. Arguably one of the most striking properties of complex systems is that conventional experimental and engineering approaches are inadequate to capture and predict the behavior of such systems. To complement the conventional experimental and engineering approaches, computer-based simulations of complex natural phenomena and complex man-made artifacts are increasingly employed across a wide range of sectors. Complex systems simulations often require considerable compute power.

The present paper describes the perspective and initial results of the QosCosGrid (Quasi-Opportunistic Supercomputing for Complex Systems) project which addresses the computationally intensive simulation of complex systems using parallel methods and grid technology. We consider the QosCosGrid approach from the perspective of the computational scientist, i.e. the user, as well as from the technology side.

Complex system simulations often use supercomputers because of the high data volume and computing requirements of the individual computations, but also because the high communication overhead between the computation tasks on individual elements. However, supercomputers are expensive to acquire and maintain and, therefore, many users do not have access to such technology. Recently, local clusters, such as Beowulf clusters and other multi-core and multi-machine systems, have become the technology of choice for many complex systems modelers. However, with the advent of flexible modeling tools, complex systems simulations have become more and more comprehensive and complex. As a result, local clusters are increasingly inadequate to satisfy the required computing and communication needs. QosCosGrid aims to address this gap by facilitating supercomputer-like performance and structure through cross-cluster computations.

In grid computing, a typical assumption is that the availability of computing resources depends on their local usage patterns. This opportunistic mode of grid computing is highly problematic for a complex system simulation. Complex system simulations are characterized by a high degree of dependency among the parallel computations that comprise the simulation. To address this problem, QosCosGrid has developed advance reservation and topology-aware methods.

2 The User Perspective

2.1 Problems of Complex System Modeling

Complex system simulations are now used across a wide range of scientific fields. However, the challenges faced by computational scientists and their demands for high performance computing differ substantially. Typically, there are many different approaches to model and simulate a concrete complex system. Each approach comes with its own requirements in terms of methods and computing technology. In an attempt to address a wide range of complex systems, we developed a classification of the computing requirements for different complex systems simulation scenarios. Key to this classification scheme is the required communication topology that reflects the element-to-element interaction characteristics of the underlying complex system.

2.2 Partitioning Complex System Simulations

The main problem in the parallelization of complex system simulations is their high demand for inter-process communication. However, parallel computing applications are most efficient when there is no or little demand for inter-node communication. In order to minimize execution time of a distributed complex system simulation, the computation needs to be partitioned such that communication between interacting components is minimized. In addition, the computational load of each partition, determining the time between communication events, needs to be balanced.

Instead of creating individual parallel solutions for every single complex system simulation, we developed a categorization of the required communication topologies (Fig. 1). The categorization scheme consists of six categories referred to as *communication templates*: ***Template T0*** is the simplest communication template. *T0* has no communication between the components, so the partitioning is only constrained by load-balancing considerations. Perfectly (or "embarrassingly") parallel applications, such as parameter sweeping, falls into the *T0* category. ***Template T5*** represents the other end of the spectrum of communication templates. Essentially, *T5* covers those complex systems for which it is difficult or impossible to determine a meaningful communication template or partitioning of the computations. ***Template T1*** describes the communication topology of a non-spatial complex system, with fixed and a priori known element interaction pattern. ***Template T2*** applies to complex systems whose element interaction patterns are changing over time. Hence the communication topology needs to be defined by some graph transition function that provides enough information for the system's efficient distribution. ***Template T3*** covers classical cellular automata simulations. In this case, the definition of a meaningful partitioning is straightforward. ***Template T4*** describes an agent-based simulations where mobile agents are embedded in a (spatial) coordinate system and have only local interactions.

Various partitioning algorithms are available for each of these communication templates. The categorization is aimed at guiding users to understand the structure of a given complex system simulation and to inform the choice of parallelization techniques. Hence, the scheme provides a useful guide for parallelizing of complex systems simulations.

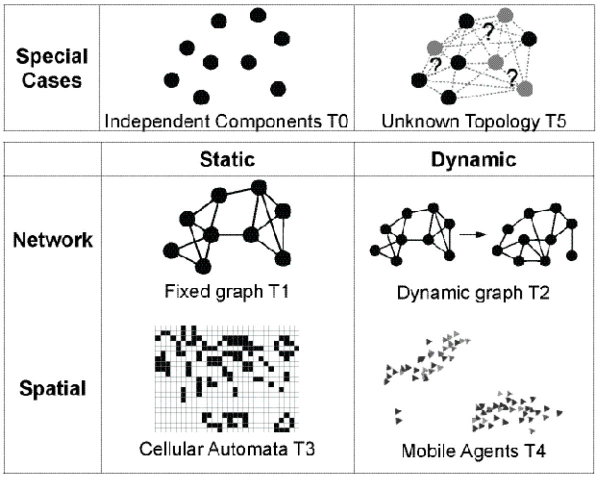


Fig. 1. Communication templates supported in the QosCosGrid system

2.3 Developing and Deploying QosCosGrid Applications

For end users, the main benefit of QosCosGrid lies in the transparency of grid-enabling complex system applications. The QosCosGrid middleware, described below, transparently handles all intricate grid aspects when it deploys complex system applications across clusters. From the perspective of the application developer, therefore, there is no difference between developing code to be deployed on a supercomputer or on a grid. Moreover, legacy parallel code can be executed on QosCosGrid without any reimplementation.

To further enhance the ease-of-use of deploying applications on the grid, a comprehensive Web interface has been developed. The QosCosGrid portal website[1] facilitates monitoring of the grid infrastructure, including bandwidth and latency, monitoring of status, usage and (advance) reservations, grid file transfer to upload applications and download results.

3 The Technical Perspective

The overall QosCosGrid architecture and the key middleware are presented in Fig. 1. Based on various middleware services, QosCosGrid components can be grouped into two levels: The *Grid* level domain (consisting of multiple organizations) and the *Administrative* level (a single organization, typically providing and sharing one or a small number of computing clusters). At one of the lowest layers there is the Local

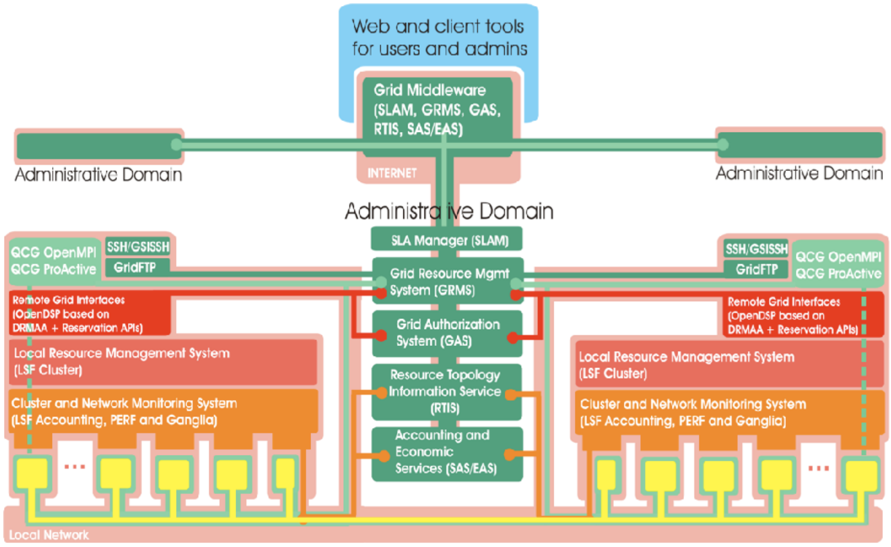

Web and client tools for users and admins
Grid Middleware (SLAM, GRMS, GAS, RTIS, SAS/EAS)
Administrative Domain
INTERNET
Administrative Domain
Administrative Domain
QCG OpenMPI
QCG ProActive
SSH/GSISSH
GridFTP
SLA Manager (SLAM)
Grid Resource Mgmt System (GRMS)
Grid Authorization System (GAS)
Resource Topology Information Service (RTIS)
Accounting and Economic Services (SAS/EAS)
Remote Grid Interfaces (OpenDSP based on DRMAA + Reservation APIs)
Local Resource Management System (LSF Cluster)
Cluster and Network Monitoring System (LSF Accounting, PERF and Ganglia)
Local Network

Fig. 2. The QosCosGrid multi-tier architecture connecting different middleware services as well as parallel application development and deployment tools

[1] http://node2.qoscosgrid.man.poznan.pl/gridsphere/gridsphere/guest/testbed/r/

Resource Management System supporting job submission, control and advance reservation (AR) mechanisms. Currently, QosCosGrid uses Platform Computing's Load Sharing Facility (LSF). We have also successfully tested the QosCosGrid middleware with the Sun Grid Engine and there are possible extensions for PBSPro or Maui. Above this layer, there is the OpenDSP service that communicates with LSF using the well-adopted standard DRMAA interface that communicates with the local queuing system. OpenDSP exposes its functionality remotely with the OGSA-BES HPC Profile compliant WS interface, and, to our knowledge, is the most efficient remote multi-user access to underlying queuing systems. On top of the OpenDSP service, there is the GRMS (Grid Resource Management Service), a main meta-scheduler service, which acts both at the Grid and Administrative level. From the end user perspective, the GRMS provides its own job description language (called Job Profile) that allows the topology requirements of various jobs, including sequential, parallel, massively parallel (with communication topology requirements) as well as workflow jobs to be defined and controlled on behalf of end users.

Below we briefly describe two main programming and execution environments, ProActive and OpenMPI, which have been successfully integrated with OpenDSP and GRMS to allow end users to perform multi-cluster job submission and control. These well-known environments have been modified according to the complex system requirements we collected. They can be distributed as powerful tools for large-scale and long-term computation analysis involving many computing clusters located in various administrative domains. The QosCosGrid middleware offers also many management features for local IT administrators responsible for sharing computing resources among many end users and their applications.

OpenMPI. The MPI (Message Passing Interface) is a leading standard in the domain of parallel scientific applications. It provides end users with both the programming interface consisting of simple communication primitives and the environment for spawning and monitoring MPI processes. A large number of implementations of the MPI standard is available (both as commercial and open source). In QosCosGrid, it was decided that MPI serves as the input for a new OpenMPI[2] distribution and we added enhancements to this implementation. Of key importance were the advance inter-cluster communication techniques that deal with firewalls and Network Address Translation. In addition, the mechanism for spawning new processes in OpenMPI needed to be integrated with QosCosGrid-developed middleware. The extended version of the OpenMPI framework was named QCG-OMPI [2] (where QCG stands for QosCosGrid).

ProActive Java. Even though enhanced OpenMPI is the primary execution environment in QosCosGrid, the existence of legacy Java applications based on the Repast Agent Simulation Toolkit [3] led to the decision to provide support for a new Java-based parallel programming environment called ProActive [4]. The ProActive library, by default, uses the standard Java RMI framework as a portable communication layer. With a reduced set of simple primitives, ProActive (version 3.9 as used in QosCosGrid) provides a comprehensive toolkit that simplifies the programming of applications distributed on local area networks, clusters, Internet grids and peer-to-peer

[2] http://www.open-mpi.org

intranets for Java-based applications. However, when we designed QosCosGrid, the standard ProActive framework did not provide any support for multi-user environments, advance reservation and cross-cluster co-allocation. To satisfy the requirements of complex system simulation applications and users, we developed extensions to the ProActive library (called QCG-ProActive) with the following goals: (1) To preserve standard ProActive library properties (i.e., allow legacy ProActive applications to be seamlessly ported to QosCosGrid). (2) To provide end users with a consistent GRMS Job Profile schema as a single document used to describe application parameters required for execution as well as resource requirements (in particular network topology and estimated execution time). (3) To prevent end users from the necessity to have direct (i.e., over SSH) access to remote clusters and machines.

Cross-site QCG-ProActive Deployment Model. ProActive deployment involves the starting of the main application and the subsequent spawning on other machines (starting ProActive Runtimes). In the context of service-level agreements and accountability, this is problematic, because it is difficult to guarantee the atomicity of such a two-phase deployment process. Application and ProActive Runtimes cannot be started at the same time because ProActive Runtimes needs to know some callback information that is used to contact the main application. Therefore, we proposed to create an external service in QosCosGrid called ProActive Node Coordinator (PNC) that helps to exchange initial arguments between the master application and ProActive Runtimes. Consequently, a local queuing system (OpenDSP with LSF) does not have to start ProActive Runtimes directly. Instead, we need to provide an appropriate wrapper script – the QosCosGrid ProActive Wrapper, which connects the PNC service and synchronizes initial data required to start ProActive Runtimes properly. Additionally, in order to support a cross-cluster ProActive deployment, before any job submission request to LSF via OpenDSP, there is an appropriate advance resource reservation call. The main difference, from the end user perspective, is that instead of a typical PAD file (ProActive Deployment Descriptor) the GRMS Job Profile is used as a language to describe ProActive application requirements and then the Job Profile is automatically converted to the corresponding PAD.

Inter-cluster QCG-ProActive Communication Mechanism. The basic ProActive Library transport layer is based on Java RMI. The RMI communication usually consumes one TCP/IP port per remote object instance and ports are randomly selected. Moreover, it is almost impossible to configure a firewall to forward the RMI traffic, which was an important issue for incorporating ProActive into QosCosGrid. To deal with this problem, the ProActive application can be configured to use the RMISSH communication protocol that simply creates on demand SSH tunnel for each outgoing RMI connection. Unfortunately, this solution does not work with sites behind Network Address Translation and it requires password-less authentication to be configured for each machine (and for every user) in the whole grid. Thus, in QosCosGrid project we proposed to use SOCKS server as the basic way to tunnel RMI traffic between clusters behind Network Address Translations and firewalls.

4 Results

The aim of QosCosGrid is to provide supercomputing-like structure and performance to cross-cluster computations for complex systems simulations. In order to assess the performance of the infrastructure provided by QosCosGrid, performance tests were conducted on a simple test bed. We ran performance tests using a cellular automaton simulation application as a benchmark application. A cellular automaton is a lattice of finite state machines, in which the state of a cell is determined by the states the neighboring cells in the previous time step [5]. Cellular automata are a widely used methodology to study spatial phenomena in physics and biology.

The regular lattice interaction topology of cellular automata makes them a prime candidate for parallelization and deployment on a supercomputer. A cellular automaton allows for simple partitioning by mapping each part to a different core or processing element. After each iteration, state information on the border cells between adjacent partitions is exchanged. As the size of a cellular automaton simulation increases, the computational cost grows quadratically, while the communication costs grow only linearly.

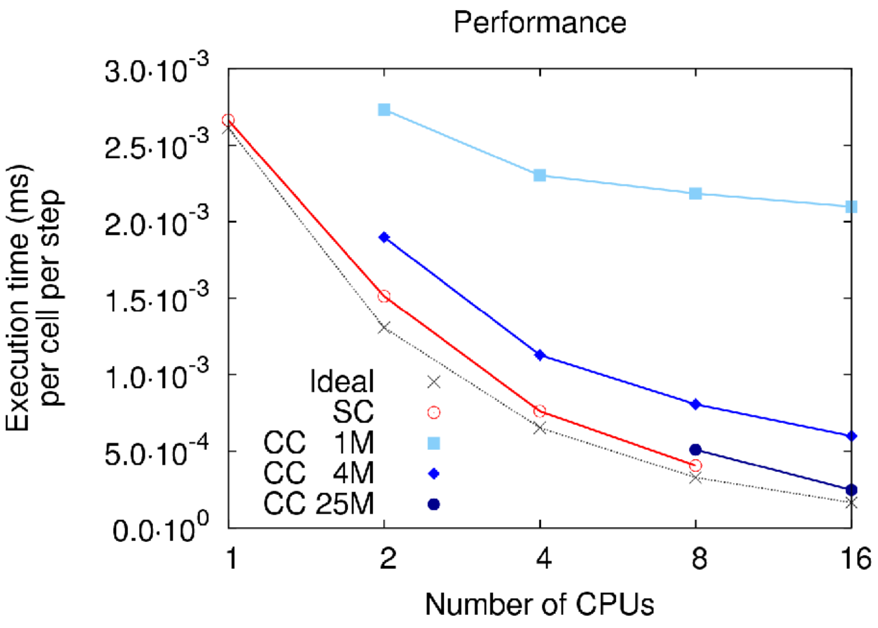


Fig. 3. Performance of a cellular automaton (CA) of increasing size (1000 x 1000, 2000 x 2000, 5000 x 5000). The line with cross-symbols shows ideal performance (execution time of non-parallel CA divided by number of cores). The line labeled SC shows single cluster runs, the CC 4M line shows cross-cluster run performance in the QosCosGrid test bed. Legend: SC = single cluster; CC = cross-cluster; 1 M = 1 million cells = 1000 x 1000; 4 M = 4 million cells = 2000 x 2000; 25 M = 25 million cells = 5000 x 5000.

Cellular automata are frequently deployed on supercomputing facilities. In this study we used cellular automata as a benchmark system to assess the performance of the QosCosGrid technology. The cellular automaton used in our study was implemented in Repast J 3.0 and parallelization was performed using QCG-ProActive. We tested the performance in a heterogeneous cross-cluster environment, and conducted experiments on two small clusters (8 cores each) located in Paris (France) and Poznan (Poland) and connected via the Internet (no dedicated network). We compared the execution time of a parallel implementation of a cellular automaton executed on a single cluster, with the execution time in a cross-cluster run.

The results (Fig. 3) describe performance data for three different cellular automaton sizes. These results confirm that parallel execution in a single cluster is highly efficient for all sizes. For relatively small cellular automaton problems (i.e., 1000 x 1000), the cross-cluster performs very poorly, as the execution time using 16 cores is similar to the performance of the non-parallel cellular automaton on a single core. As the size of the cellular automaton increases, however, the performance of cross-cluster execution approaches the performance in a single cluster. For the largest cellular automaton in the performance test (5000 x 5000), the performance of the cross-cluster execution becomes almost indistinguishable from the single cluster performance.

These preliminary results show that parallel complex systems simulations with a relatively high computation/communication ratio can be meaningfully deployed on the QosCosGrid grid infrastructure.

5 Discussion

QosCosGrid is an end-to-end solution that is already being used by scientists for compute intensive parallel applications in the field of complex systems modelling. The advantage of QosCosGrid is that it requires no dedicated network connections or specific configurations. It is a multi-user environment that efficiently controls access to computing resources in different administrative domains. It is specifically designed for non-trivial parallel computations using Open-MPI with C/C++ and Fortran, or ProActive for Java-based legacy applications. Additional features of interest to scientists include its user-friendly Web-based user interface and its reusable application schemas or template categories based on ProActive.

QosCosGrid is aiming to provide a high quality of service (for users) equivalent to that of a dedicated supercomputing facility. To achieve this, QosCosGrid provides services such as co-allocation and advance resource reservation [6], which are very difficult to provide in dynamic grid environments. Recent efforts in these areas include that by Elmroth and Tordsson for the NorduGrid/ARC middleware [7] and the work by Kyriazis and colleagues [8] who concentrate on orchestrating grid resources in order to support application workflows. More established solutions are provided by Condor-G [9], and Nimrod-G [10]. After a careful review of existing open solutions and standards, QosCosGrid has based its services on top of exiting third-party software. In addition, the QosCosGrid Gateway is a key feature for high-quality of service as it provides essential support for both users and administrators.

Mateos *et al.* [11] provide a useful survey that compares a number of different approaches to grid-enabling applications. They discuss a number of other Java tools used in grids, apart from ProActive, such as Satin [12], which is based on Ibis [13], and “grid-aspecting” [14], which is based on AspectJ. The difference between these Javas and ProActive is in the degree of granularity and the amount of modification required to the application source code. Both ProActive and Satin require more extensive modifications to the source than grid-aspecting, but an advantage of these modifications is that applications can make more sophisticated and efficient use of grid resources.

There are other projects which have aims that are similar to those of the QosCosGrid project, these include EGEE [15], HPC4U [16] and DEISA [17]. However, these projects do not provide all the functionality of QosCosGrid. EGEE does not support all of QosCosGrid's quality-of-service components such as advance reservation and check-pointing. HPC4U concentrates on parallel executions performed within clusters as opposed to QosCosGrid's cross-cluster execution environment. DEISA, the Distributed European Infrastructure for Supercomputing, is a grid of supercomputers suitable for petascale applications whereas QosCosGrid is a more “humble” system designed to enable distributed clusters to be combined into a resource with compute power similar to a single supercomputer.

6 Conclusions

This paper describes how QosCosGrid enables clusters in different administrative domains to be welded (virtually) into a single powerful compute resource which we call a *quasi-opportunistic supercomputer*. We outlined the middleware that we have developed to achieve this and although QosCosGrid provides extensive support via OpenMPI for parallel C/C++ and Fortran applications, here we have mainly focussed on the modifications we made to ProActive Java for supporting inter-cluster communications and dealing with firewalls and Network Address Translations. Our results, which are based on an ecological simulation using parallelized cellular automata, demonstrate the feasibility of running non-trivial parallel simulations across administrative domains located in different European countries. For large simulations there is only a minor reduction in performance when running an inter-cluster simulation compared to a single cluster simulation. QosCosGrid is a largely open-source project which is due to be completed in mid-2009.

Acknowledgments. The work described in this paper is supported by EC grant QosCosGrid IST FP6 STREP 033883.

References

1. Special edition: Complex Systems. Science 284, 5411, 1–212 (1999)
2. Coti, C., Herault, T., Peyronnet, S., Rezmerita, A., Cappello, F.: Grid Services for MPI Cluster Computing and the Grid. In: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2008), pp. 417–424 (2008)
3. North, M.J., Collier, N.T., Vos, J.R.: Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Transactions on Modeling and Computer Simulation 16(1), 1–25 (2006)

4. Caromel, D., Delbe, C., di Costanzo, A., Leyton, M.: ProActive: an Integrated Platform for Programming and Running Applications on Grids and P2P systems. Computational Methods in Science and Technology 12(1), 69–77 (2006)
5. Farmer, J.D., Toffoli, T., Wolfram, S. (eds.): Cellular Automata. Elsevier, Amsterdam (1984)
6. Kravtsov, V., Schuster, A., Carmeli, D., Kurowski, K., Dubitzky, W.: Grid-enabling complex system applications with QosCosGrid: An architectural perspective. In: Proceedings of The International Conference on Grid Computing and Applications (GCA 2008), Las-Vegas, USA (2008)
7. Elmroth, E., Tordsson, J.: Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. Fut. Gen. Comput. Sys. 24(6), 585–593 (2008)
8. Kyriazis, D., Tserpes, K., Menychtas, A., Litke, A., Varvarigou, T.: An innovative workflow mapping mechanism for Grids in the frame of Quality of Service. Fut. Gen. Comput. Sys. 24(6), 498–511 (2008)
9. Frey, J., Tannenbaum, T., Mirnon, L., Foster, I., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing 3, 237–246 (2002)
10. Abramson, D., Buyya, R., Giddy, J.: A computational economy for grid computing and its implementation in the Nimrod-G resource broker. Fut. Gen. Comput. Sys. 18(8), 1061–1074 (2002)
11. Mateos, C., Zunino, A., Campo, M.: A survey on approaches to gridification. Softw., Pract. Exper. 38(5), 523–556 (2008)
12. Wrzesinska, G., Maassen, J., Verstoep, K., Bal, H.E.: Satin++: Divide-and-Share on the Grid. In: Second IEEE International Conference on e-Science and Grid Computing (e-Science 2006) (2006)
13. van Nieuwpoort, R.V., Maassen, J., Wrzesinska, G., Hofman, R., Jacobs, C., Kielmann, T., Bal, H.E., Henri, E.: Bal: Ibis: a Flexible and Efficient Java based Grid Programming Environment. Concurrency and Computation: Practice and Experience 17(7-8), 1079–1107 (2005)
14. Maia, P.H., Mendonça, N.C., Furtado, V., Cirne, W., Saikoski, K.: A process for separation of crosscutting grid concerns. In: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC 2006, Dijon, France, April 23 - 27, 2006, pp. 1569–1574. ACM, New York (2006)
15. Gagliardiand, F., Jones, B., Grey, F., Bgin, M.-E., Heikkurinen, M.: Building an infrastructure for scientific grid computing: Status and goals of the EGEE project. In: Philosophical Transactions A of the Royal Society: Mathematical, Physical and Engineering Sciences, vol. 363(833), pp. 1729–1742 (2005)
16. Hovestadt, M.: Operation of an SLA-aware grid fabric. IEEE Trans. Neural Networks. 2(6), 550–557 (2006)
17. Lederer, H., Hatzky, R., Tisma, R., Bottino, A., Jenko, F.: Hyperscaling of plasma turbulence simulations in DEISA. In: Proceedings of the 5th IEEE Workshop on Challenges of Large Applications in Distributed Environments, Monterey, California, pp. 19–26 (2007)

Geostatistical Computing in PSInSAR Data Analysis

Andrzej Lesniak and Stanislawa Porzycka

AGH University of Science and Technology Krakow, Poland
lesniak@agh.edu.pl, porzycka@agh.edu.pl

Abstract. The presented paper describes the geostatistical analysis of PSInSAR data. This analysis was preceded by short description of PSInSAR technique. The geostatistical computations showed in this article were performed with the *R* open-source software containing the *gstat* package. The analysis contains variograms computing (directional variograms) and ordinary kriging interpolation. The computationally costly problems in geostatistical analysis of PSInSAR data were discussed.

Keywords: geostatistics, ground deformations, interferometry, kriging, PSInSAR.

1 Introduction

The computations in geosciences frequently require working with different types of data. Nowadays the satellite data are used very often. Among them we can distinguish radar and multispectral images. Both types derive information for each pixel of the images. Quite often, geoscientists have to analyze these remote sensing data in relation to ground measurements, which are usually performed for irregular and rare grid due to economical and physical constraints. Comparing large remotely sensed data with ground measurements is not only an extremely important task, but also a very computationally costly challenge. An example of a computationally intensive processing component in geographic data analysis is kriging, which is a geostatistical method of interpolation.

This paper presents the analysis of PSInSAR data. PSInSAR method is dynamically developed branch of satellite radar interferometry. Nowadays many institutions have implemented the PSInSAR technique to monitor ground deformations. This technique cannot constitute an independent tool to study ground displacements and PSInSAR data have to be joined with other measurements. The analysis of PSInSAR data, which are huge data sets, meets several computational problems. In this paper the analysis of PSInSAR data was performed with the use of geostatistical methods. The main goal of this analysis was to interpolate values of ground displacements at unmeasured locations to produce maps, which can be easily analyzed together with satellite images or with interferograms. Performed interpolation gives us the opportunity to do a useful comparison between different datasets. This work points the main computationally costly problems in geostatistical analysis of PSInSAR data.

2 PSInSAR Data Set

Scientists from the Politecnico di Milano (POLIMI) elaborated the PSInSAR (Permanent Scatterer Interferometry SAR) technique in the nineties of the 20th century. This method exploits sets of dozens SAR images in order to detect small (not bigger than several centimeters per year), long period ground deformations [1]. SAR (Synthetic Aperture Radar) is a form of radar mounted on the satellite. It emits its own microwave radiation towards the surface of the Earth and records the amplitude and phase of the signal, which returns to the radar antenna (each pixel of the radar image contains information about the phase and amplitude of backscattered signal). PSInSAR technique derives information about ground deformations only for PS points. PS points are stable radar targets, which means that PS points have time stable amplitude and phase in all exploited radar images. These stable radar targets correspond very well with man-made features on the ground, such as buildings, bridges, viaducts and etc., therefore the density of PS points is much higher in urban areas (even more than 1000 PS/km^2) than it is in rural areas. PSInSAR method provides information about ground displacements for large areas of interest, even exceeding 10 000 km^2. The spacing of PS points is usually very irregular. PSInSAR technique uses archival images, dating back to 1992, and giving us the opportunity to reconstruct previous ground deformations. This method enables to detect displacements with average annual rate equal to 0.1 mm/yr. Despite the fact that PSInSAR technique cannot be an independent tool for ground movements monitoring, it complements considerably the conventional leveling and GPS surveying.

PSInSAR data, which have been presented in this paper, describe small, long-lasting ground displacements, which occurred in the Upper Silesian Coal Basin (Southern Poland) in the years between 1992 and 2003. In this region the intensive coal exploitation has been carried on for more than two hundred years. This exploitation and complicated geological structure (a lot of faults) makes this area particularly endangered with terrain deformations. PSInSAR data for Upper Silesian Coal Basin were obtained as a result of 79 SAR images processing. These radar images were performed by ESA's satellites (ERS-1, ERS-2 and ENVISAT). In the studied region, which covers more than 1200 km^2, about 120 000 PS points were identified (Fig. 1). For each of them the average annual motion rate (mm/year) and value of coherence were calculated. For 30 000 of these PS points the values of monthly, relative ground deformation were also determined. Locations of PS points correspond very well with the land development. In this region there are also areas without PS points. These areas represent agricultural regions, forests and areas with strong ground displacements caused directly by mining activity (in this last case areas without PS points are located usually exactly above exploitation parcels) [2]. For the Upper Silesian Coal Basin the subsidence phenomenon is characteristic. The values of average annual motion rates in this region range from -39 mm/yr to 25 mm/yr. In order to explain the origin and mechanism of ground deformations in this studied region the PSInSAR data have to be analyzed together with geological, hydrogeological and mining data. This analysis has to be preceded by exploratory PSInSAR data study and interpolation of displacements at unobserved locations.

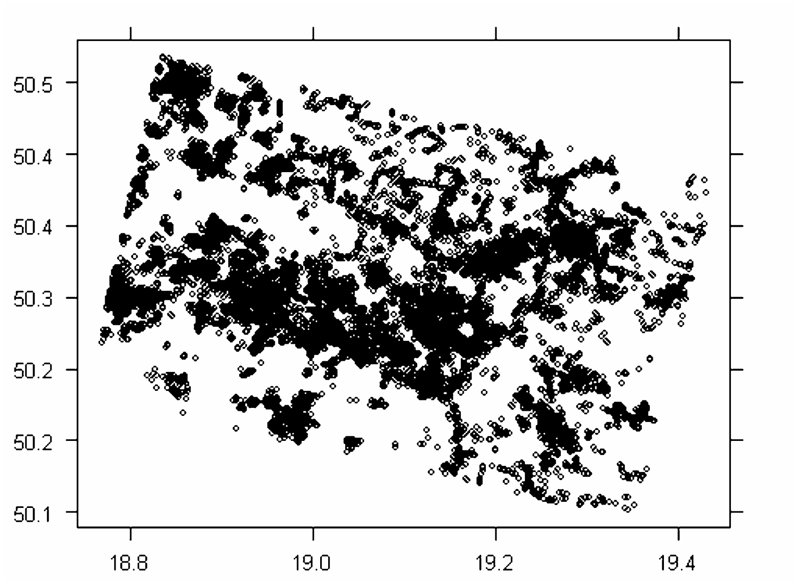


Fig. 1. Location of 30 000 PS points in the area of Upper Silesian Coal Basin (southern Poland)

3 Geostatistical Analysis of PSInSAR Data

The PSInSAR data analysis was performed with the use of geostatistical methods. Geostatistics is a subset of statistics specializing in analysis and interpretations of geographically referenced data [3]. In geostatistics a spatial autocorrelation among sample data is described. This autocorrelation is modeled by a semivariogram, which plots the semivariance as a function of distance. The semivariogram (empirical, experimental) can be estimated from *Nh* sample data pairs *z(si)* and *z(s$_i$+h)* for a number of distances (or distance intervals) h_j by Eq.(1).

$$\gamma(\tilde{h}_j) = \frac{1}{2N_h} \sum_{i=1}^{N_h} (z(s_i) - z(s_i + h))^2, \forall h \in \tilde{h}_j \tag{1}$$

The semivariograms provide insight into the spatial structure of a random process. One of the main goals of geostatistics is to predict values of variable at unobserved locations (in space or in time). Kriging is the geostatistical method of prediction. It is based on the theory of the regionalized variables [4]. Interpolation of value of variable at an unmeasured location is based on observations of its value at nearby locations. A standard version of kriging is called ordinary kriging. The predictions are made as in Eq.(2).

$$\hat{z}_{OK} = \sum_{i=1}^{n} w_i(s_0) z(s_i) = \lambda_0^T z \tag{2}$$

where the λ_0^T is a vector of kriging weights (*wi*) and *z* is the vector of *n* observations of primary locations. The values of kriging weights should reflect the true spatial autocorrelation structure and they are given by ordinary kriging equation system Eq.(3):

$$\begin{bmatrix} w_0(s_0) \\ \vdots \\ w_n(s_0) \\ \varphi \end{bmatrix} = \begin{bmatrix} \gamma(s_1,s_1) & \cdots & \gamma(s_1,s_n) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \gamma(s_n,s_1) & \cdots & \gamma(s_n,s_n) & 1 \\ 1 & 1 & \cdots & 0 \end{bmatrix}^{-1} \begin{bmatrix} \gamma(s_0,s_1) \\ \vdots \\ \gamma(s_0,s_n) \\ 1 \end{bmatrix} \tag{3}$$

where φ is called *Langrange* multiplier. In addition to the estimation we can also calculate the prediction variance (variance of the prediction error) Eq.(4):

$$\sigma_{OK}^2 = C - \sum_{i=1}^{n} w_i(s_0)C(s_0,s_i) + \varphi \tag{4}$$

where $C(s_0,s_i)$ is the covariance between the new location and the sampled point pair and *C* is a sill (upper bound) of semivariogram. Values of prediction variance derive information about quality of used kriging model.

Geostatistical analysis of PSInSAR data for the area of Upper Silesian Coal Basin was performed using *R* with *gstat* package. *R* is a language and environment for statistical computing and graphics. It provides methods for advanced statistical analysis. *Gstat* is a package, which derives functions for geostatistical analysis. The PSInSAR data analysis was done for 1240 randomly selected PS points and includes four main steps (excluding the exploratory data analysis): semivariogram estimation, kriging, calculation of prediction variance and cross-validation. All calculations were performed on a PC.

In the first part of the analysis the values of empirical (experimental) semivariograms were computed. In order to check how the data's variation depends on the relative orientation of data locations the four directional semivariograms were calculated. They were performed for directions: 0, 45, 90 and 135 (where 0 is North and 90 is East) (Fig. 2). In the next step of analysis the isotropic semivariogram was created (Fig. 3). The obtained isotropic semivariogram has a shape suggesting a spherical model. This model was adjusted by weighted least-squares (Fig. 3). In case of selected PSInSAR data the distance at which the semivariogram reaches the sill (range) is equal 12.2 km and beyond this range no correlation exists between two values of ground displacements at PS points.

The isotropic semivariogram estimation is a computationally time-consuming task but it has to be executed only once per data set. The same situation holds also for semivariogram theoretical model fitting. The semivariograms calculation can cause more problems in case of anisotropy that can be modeled by defining range ellipse [5]. In this case several directional semivariograms have to be calculated and then for each of them the theoretical models have to be fitted. Figure 4 shows the relation between numbers of used PS points and semivariogram computational times. The relation is distinctly non-linear. In case of large PSInSAR dataset necessity of calculation more than one semivariogram causes meaningful increase of computational time.

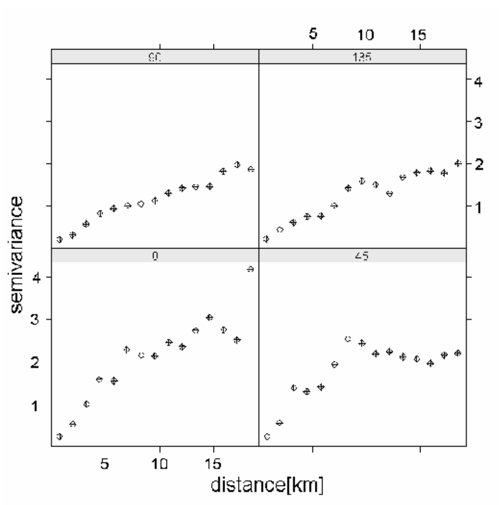


Fig. 2. Directional semivariograms for four directions (0 is North, 90 is East)

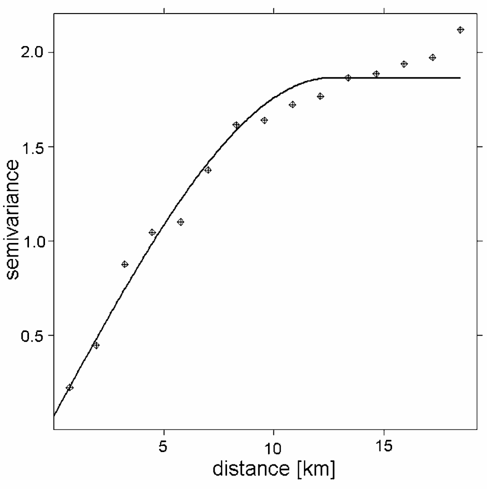


Fig. 3. Experimental isotropic semivariogram with fitted theoretical model

The goal of the next step in the geostatistical analysis of PSInSAR data was to predict values of ground deformations at unmeasured locations. This task was performed using ordinary kriging method. For PSInSAR data it is important to predict the values of variable for very dense grid. It is essential when the results of interpolation are used to study the stability of individual buildings. In case of PSInSAR data, kriging computations are hindered because of location of PS points, which is very irregular. In this work the values of subsidence were predicted for the grid with only 20000 nodes. The results of kriging are presented in the Fig. 5.

Kriging is an example of computationally intensive method because it requires the solution of a large linear system for each grid node. In the case of large datasets analysis (like PSInSAR dataset) kriging it is too computationally demanding to be run on a PC or low performance computing platform. The kriging produces the best results when the largest possible number of known points is used to predict values in no measured location [6]. This is the most expensive option. In case of PSInSAR data analysis even several thousand points can be used to estimate the variable for one grid node.

The computational time of kriging proceeding also increases when the interpolation is done for the dense grid. In this work the computational times for kriging proceeding were measured in the *R*. In the first case the elapsed time was measured in relation to the number of grid nodes where the values of ground deformations were predicted. In this case the number of PSInSAR data was constant

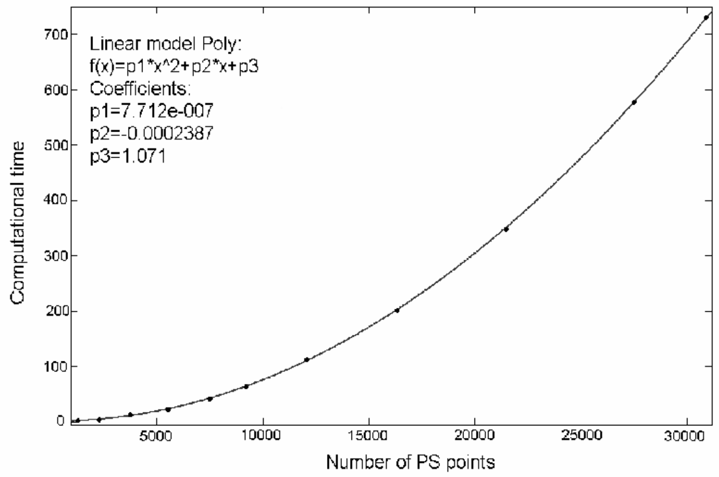


Fig. 4. Timing results for isotropic semivariogram algorithm for different numbers of PS points

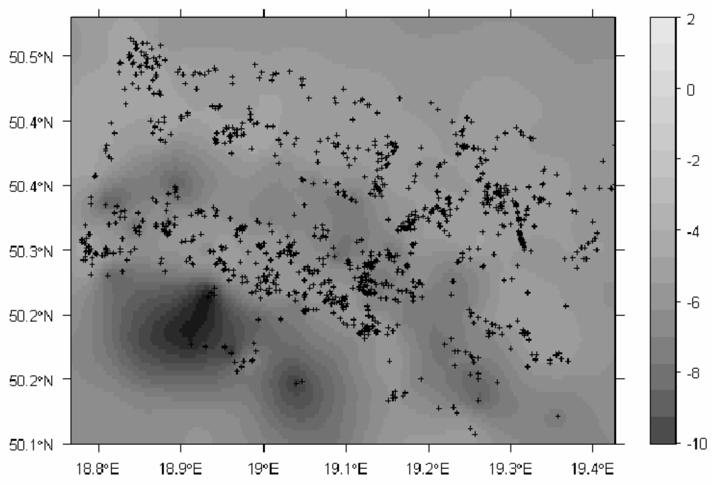


Fig. 5. Ordinary kriging output for average annual motion rate [mm/yr]

and equal to 1240 PS points. For the studied area five regular grids were prepared. The numbers of nods of these grids were corresponding to: 1250, 5000, 20000 and 80000 nodes. As it can be seen in the Fig. 6 the relation between the computational time and the number of grid nodes is linear e.g. double increase of grid nodes causes double increase of kriging computational time.

In the second case the number of grid nodes was constant and equal to 5000 nodes. In this part of work the relation between kriging computational times and numbers of PS points (used to interpolate values at grid nodes) was studied. It should be underlined that the spacing of PS points is irregular and the subsets of PS points were selected randomly. The number of PS points was changed from 83 to 2395 points. In this case the relation between the computational time and the number of PS points is non-linear (Fig. 7). It can be evaluated that for 120 000 PS points the computational time equals about 5 days and 13 hours. In order to reduce this time the kriging algorithm can be performed in parallel environment [7].

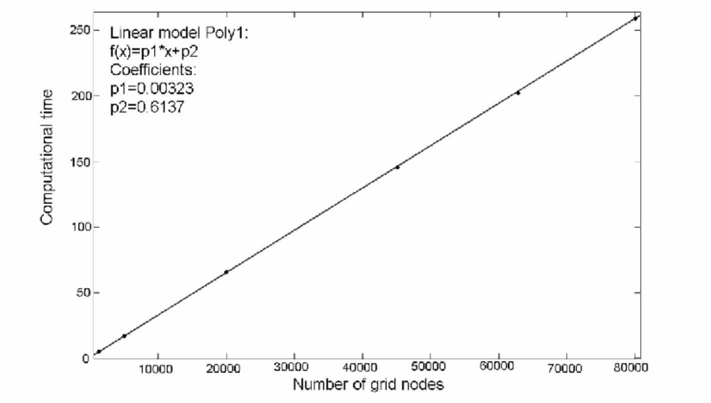


Fig. 6. Timing results for kriging algorithm for different numbers of grid nodes

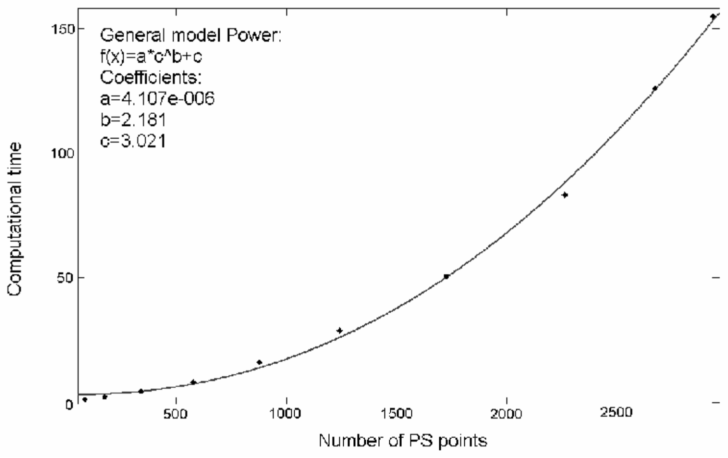


Fig. 7. Timing results for kriging algorithm for different numbers of PS points

In the third part of analysis the map of prediction variance was performed (Fig. 8). The ability of kriging to produce this kind of map is what separates it from other spatial interpolation methods.

In the last part of the geostatistical analysis the leave-one-out cross validation method was used to pinpoint the most problematic PS points. This algorithm of leave-one-out cross validation is a very computationally costly procedure. In this method the value of variable for each individual point is assessed against the whole data set. Each data point is visited and the prediction is done with kriging method, but without using the observed value. Fig. 9 presents the timing results for leave-one-out cross validation algorithm for different numbers of PS points. The number of PS points was changed from 83 to 1241 points. The relation between the computational time and the number of PS points is non-linear.

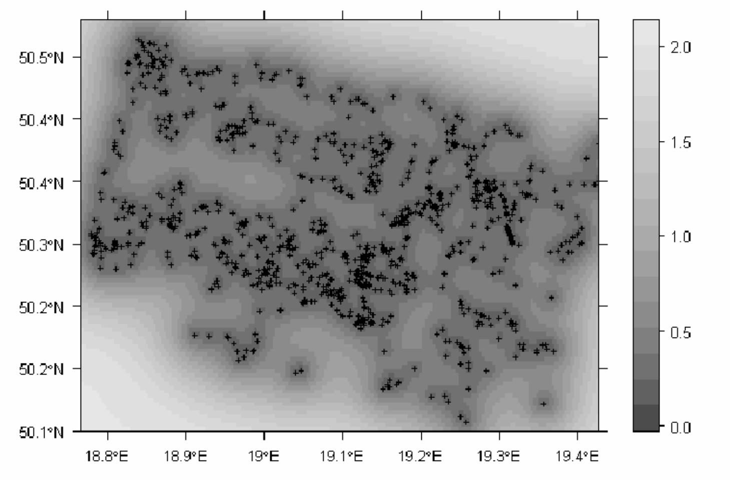


Fig. 8. Map of prediction variance

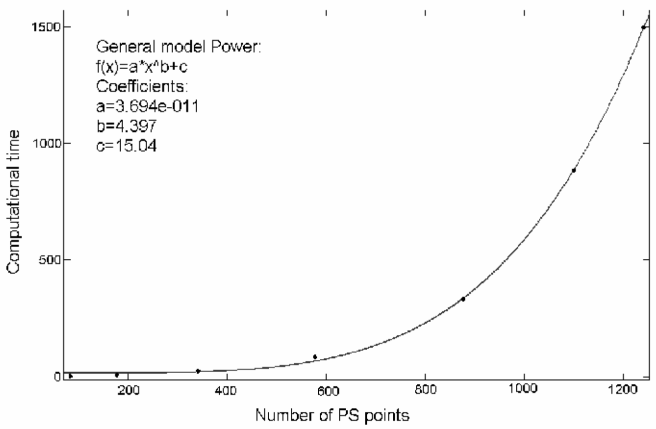


Fig. 9. Timing results for cross validation algorithm for different numbers of PS points

4 Conclusions

The geostatistical analysis of PSInSAR data gives good results, yet it is a very computationally costly procedure. In this work the analysis was performed only for 1240 PS points selected randomly from the dataset which includes 120 000 PS points. Kriging is the most computationally costly task in geostatistical analysis of ground deformation. This task is also crucial because kriging results constitute the base for the geological interpretations. The maps of kriging output and prediction variance obtained in this work and complemented by different kind of data (geological, hydrogeological, mining) enable to determine the relations between values of subsidence and mining activity and geological structure of studied region. The interpolation of ground displacements for very dense grid give us also opportunity to monitor stability of individual objects on the ground e.g. buildings.

For small datasets the geostatistical analysis can be run on a PC but in case of the whole PSInSAR dataset it is necessary to use high performance computing platform or distributed architectures. In order to perform the geostatistical analysis for all 120 000 PS points the parallel kriging algorithm has to be prepared. Designing this algorithm the very strong irregularity in data locations has to be taken into consideration.

References

1. Ferretti, A., Prati, C., Rocca, F.: Permanent Scatterers in SAR Interferometry. IEEE Transaction on Geoscience and Remote Sensing 39(1), 8–20 (2001)
2. Lesniak, A., Porzycka, S.: Environment monitoring using satellite radar interferometry technique (PSInSAR). Polish Journal of Environmental Studies 17(3A), 382–387 (2008)
3. Goovaerts, P.: Geostatistics for Natural Resources Evaluation. Oxford University Press, New York (1997)
4. Weckernagel, H.: Multivariate Geoststistics. Springer, Heidelberg (1995)
5. Bivand, R.S., Pebesma, E.J., Gomez-Rubio, V.: Applied Spatial Data Analysis with R. Springer, New York (2008)
6. Border, S.: The use of indicator kriging as a biased estimator to discriminate between ore and waste. Applications of Computers in the Mineral Industry, University of Wollongong, N.S.W (1993)
7. Kerry, K.E., Hawick, K.A.: Kriging Interpolation on High-Performance Computers. In: Bubak, M., Hertzberger, B., Sloot, P.M.A. (eds.) HPCN-Europe 1998. LNCS, vol. 1401, pp. 429–438. Springer, Heidelberg (1998)

Improving the Scalability of SimGrid Using Dynamic Routing

Silas De Munck, Kurt Vanmechelen, and Jan Broeckhove

University of Antwerp,
Department of Computer Science and Mathematics,
Middelheimlaan 1, 2020 Antwerp, Belgium
silas.demunck@ua.ac.be

Abstract. Research into large-scale distributed systems often relies on the use of simulation frameworks in order to bypass the disadvantages of performing experiments on real testbeds. SimGrid is such a framework, that is widely used and mature. However, we have identified a scalability problem in SimGrid's network simulation layer that limits the number of hosts one can incorporate in a simulation. For modeling large-scale systems such as grids this is unfortunate, as the simulation of systems with tens of thousands of hosts is required. This paper describes how we have overcome this limitation through more efficient storage methods for network topology and routing information. It also describes our use of dynamic routing calculations as an alternative to the current SimGrid method which relies on a static routing table. This reduces the memory footprint of the network simulation layer significantly, at the cost of a modest increase in the runtime of the simulation. We evaluate the effect of our approach quantitatively in a number of experiments.

Keywords: Grid Computing, Grid Simulation, Scalability, SimGrid, Routing Algorithm, Boost Graph Library.

1 Introduction

Distributed and parallel processing techniques are common today in a wide range of applications. The increasing scale and complexity of distributed applications and systems necessitates research into more scalable and efficient algorithms and techniques for e.g. resource management and job scheduling. The evaluation of new algorithms on real testbeds is however impeded by their limited flexibility, controllability and availability. In addition, the costs for building and configuring large-scale testbeds are high.

For this reason, researchers turn to simulation to evaluate new algorithms and techniques, especially during the initial phases of development. A widely used simulation toolkit in this regard is SimGrid [1]. SimGrid is a toolkit providing functions for the simulation of distributed applications in heterogeneous distributed environments. It thereby targets platforms that range from a simple network of workstations to large-scale computational grids.

However, we have found SimGrid to have a scalability problem that impedes the simulation of large-scale systems such as grids. Currently, the size of the simulated networks is mainly limited by system memory. This memory limit is reached at roughly 4000 hosts on a machine with $8\,GB$ of memory. Although scaling beyond this limit is possible through virtual memory, this results in extensive swapping which cripples the simulator's performance. The large memory requirements of SimGrid are due to the memory-intensive manner of storing the routing information that is used in the simulated network. Another but less important problem is due to the structure of the *platform (description) files* that describe the network topology in SimGrid. These files are larger than necessary, resulting in significant startup costs of the simulation and decreased manageability of those files.

This paper introduces methods for improving the scalability of the SimGrid simulator. The network representation and the traffic routing functions in particular will be involved. SimGrid's original route information storage and lookup methods will be briefly discussed. Subsequently, different methods to improve them will be described and evaluated using a proof of concept implementation.

2 Scalability Issues

This section gives an overview of the scalability issues that we have identified in SimGrid. As mentioned before, the maximum attainable number of hosts in a simulation is limited by the available memory in the system. We take a look at the way SimGrid processes routing and topology information and how this influences the memory usage of the simulator.

2.1 SimGrid Simulation Infrastructure

The low-level network simulation in SimGrid is performed by the *SURF*-module. *SURF* provides the core functionalities to simulate a distributed system and is also responsible for the parsing and processing of the platform files. Another module, called *SIMIX*, is an intermediate layer between the low-level *SURF*-module and the high-level *MSG* user API. *SIMIX* uses *deployment (description) files* to describe a simulation scenario by assigning *processes*, which contain user-defined logic, to hosts. A typical SimGrid simulation run will first process the platform and deployment description files and then launch the simulation.

2.2 The Platform Files

SimGrid platform files are used to describe the topology of the network to be simulated. These files contain an XML-based description of hosts, network links and routing paths. The specified entities have properties like bandwidth or latency for links, and processing power for hosts. Routing information is statically specified by a list of links forming the path between each pair of hosts in the network. Note that the specified path from source to destination does not necessarily constitute the shortest path, the route can consist of any chain of links.

A recent revision of the platform file XML schema [2] has made it possible to specify a group of hosts using a *cluster* element. A cluster element in the platform file contains a template host configuration for the cluster hosts, that will be expanded to real hosts by SimGrid during the processing of the platform file. This functionality reduces the amount of redundant information and thus the size of the platform file. Specifying routes between clusters is also supported and will result in the generation of routes between the hosts in the source and destination clusters during the parsing process. This means that the simulation code itself has no knowledge about clusters, and that is the cause of a first inefficiency in the network model. Since the simulator only has knowledge about hosts, it has to know the specific routes from and to every host in the network.

Although the inclusion of the cluster element has already led to a significant reduction in the size of platform files, the need to fully specify the routes between all clusters in the platform files still results in rather large files. Furthermore, because *SURF* has no notion of clusters and routes between clusters, all the individual hosts of a cluster and the associated routes between them need to be expanded to an in-memory representation.

2.3 The Simulation Datastructures

During the parsing of the platform file, *SURF* stores the specified hosts and routes, as lists of links in a `xbt_dict_t`[1]. When the parsing is finished, the routing information is converted to one large two-dimensional array of size $n \times n$ (with n the number of hosts) containing the network link lists. Each array-element contains a list of pointers to the actual link data structures that make up the route from one host to another. This array contains all topology and routing information necessary to perform the simulation.

The use of the routing array has a substantial impact on the memory usage of the simulator. A significant amount of information in the route array is redundant, as the hosts of the same cluster have identical routes. Nevertheless, the advantage of this method is that the lookup of a specific route is very fast as it merely involves access to memory by direct indexing. The time and space complexity of the route lookup algorithm are respectively $O(1)$ and $O(n^2 l)$ where n is the number of hosts and l is the average number of links in a routing path.

3 Improving Scalability

As discussed before, the number of hosts in SimGrid is limited by memory. The root of this problem lies in the way the routing information is kept in memory and in the platform files. They both contain redundant routing information by defining all paths between all hosts, even though a specification of the links from

[1] `xbt_dict_t`: SimGrid uses its own dictionary data structure `xbt_dict_t` from the *XBT*-module which associates a `char*` with any `void*` data.

a host to its neighbours is sufficient to determine the routes between all pairs of hosts. By eliminating this redundancy in the routing table, scalability should improve significantly. We have implemented in SimGrid a number of routing algorithms to calculate routes dynamically. This eliminates the need for the extensive all-to-all routing table, reducing the memory footprint and the size of the platform files. In this section, we present a number of routing algorithms that calculate the required routing path at runtime. This consequently reduces the amount of memory needed and the size of the platform files.

Table 1 contains formulas that approximate the routing tables' memory usage for the different algorithms, where n is the number of hosts, l the average number of links in a path, m the average number of outgoing links per host, and c the number of cached entries (the maximum is n).

3.1 Floyd's Algorithm

We have implemented, in SimGrid, the Floyd-Warshall [3] algorithm for finding the shortest path between two hosts in the network. It uses an adjacency matrix that describes the network topology and it produces a predecessor matrix that holds the destination hosts' predecessor in the path between a pair of hosts, and a cost matrix that contains the total cost of that path. The cost and predecessor matrices are calculated before the start of the simulation itself. When this procedure is completed, the list of links composing the route can be easily constructed using the predecessor matrix.

The use of this algorithm results in a significant reduction in memory use. The size of route information data now only depends on the number of hosts, as opposed to the original algorithm where the length of the route path is an important factor. This reduction comes at a cost however, as the algorithm requires more time to initialize before the simulation starts. The route lookups during the simulation, compared to the current SimGrid implementation, are only slightly slower. The time and space complexity of the route initialization algorithm are now $O(n^3)$ and $O(n^2)$ respectively. The lookup of the route link list takes only $O(l)$ where l is the average number of links in a routing path.

Table 1. Routing table memory usage approximation formula's

Method	Mem. Usage Formula	Information stored in memory
Original	$n^2 l \times size_{ptr}$	list of (pointers to) links per host pair
Floyd	$n^2 \times (size_{double} + size_{int} + size_{ptr})$	path cost, predecessor ID of path and link-pointer per host pair
Dijkstra	$n \times (size_{ptr} + size_{int}) + nm$	host ID, adjacent host IDs and link-pointers
Dijkstra w. cache	$n \times (size_{ptr} + size_{int}) + nm + nc \times size_{int}$	host ID, adjacent host IDs, link-pointers and cached predecessor IDs of paths from source host

3.2 Dijkstra Shortest Path Algorithm

A standard Dijkstra shortest path algorithm [4] can be used from within the simulation itself, which eliminates the need for initialization of the routing table, and it will further reduce the memory footprint. For the implementation of this algorithm we have used the C++ Boost Graph Library (BGL) [5]. The algorithm produces a predecessor list for all routing paths starting from one source host to every other host in the network, based on a adjacency list. The reduction in memory usage is due to the use of an adjacency list instead of a matrix to represent the network as a graph.

The Dijkstra shortest path algorithm yields a significant reduction of the memory needed. The space complexity of this algorithm is $O(nm)$, with m the average number of outgoing links per host. This algorithm results in a linear instead of quadratic increase in memory usage as a function of the number of hosts. The time complexity in this case adds up to $O(n\,log\,n)$ for each route lookup.

3.3 Dijkstra with Caching

The Dijkstra algorithm is a substantial improvement when it comes to memory usage, but it has a large impact on runtime performance. To improve this situation, we have used a cache for the calculated predecessor array. All routes from one source host to all other hosts are cached at the first lookup of a path from the source host. The space complexity then becomes $O(nm + cn)$, with m the average outgoing links and c the cache size. Time complexity is still $O(n\,log\,n)$ for the first calculation of a route and $O(log\,c)$ for all succeeding route lookups from the cache, where c is the size of the cache. A C++ Standard Template Library (STL) map [6], is used for the cache. The effect of this cache on actual memory use and runtime performance depends on the simulated scenario.

3.4 Necessary Topology Changes

Implemention of the aforementioned algorithms requires some changes to SimGrid, as SimGrid allows one to describe a route as a chain of links in the platform files where the links do not necessary have real hosts (or other hops) in between them. Our route calculation algorithms are not capable of working with such a route description, because they calculate a routing path as a composition of single links between hosts. The specification of the routes as a chain of links does not allow one to derive the network topology graph, as two hosts in the same cluster can be configured to have a completely different route to the same destination.

Another problem is that SimGrid implicitly creates links when a cluster is specified. To still be able to use the cluster declaration syntax in the platform files, the generation of links for clusters in *SURF* was altered. SimGrid currently only creates a backbone link in a cluster, connecting all of its hosts. The backbone link should then be used in the platform files to link the cluster hosts to a gateway

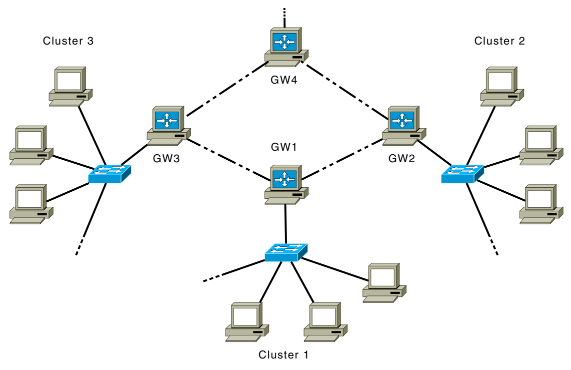


Fig. 1. New cluster network topology

connecting to the rest of the network. This way, all incoming and outgoing traffic has to pass through the cluster's gateway host. To perform a fair comparison against the original SimGrid routing method, this network topology change is also used with the original routing scheme. Figure 1 shows this new topology scheme graphically.

4 Experiments

We compare the algorithms of the previous section to the original approach used in SimGrid. We evaluate the differences in performance, memory usage and disk space requirements. Our evaluation has been carried out with the `masterslave_forward` example that comes with the SimGrid source code. This example simulates a number of *master* hosts sending tasks to other *slave* hosts. Platform files of various sized networks with their corresponding deployment files were generated. The deployment files define 1 host for each cluster that functions either as master or as a slave, and each master has 4 other slave hosts. Each master distributes 20 "tasks" among its slaves. The tests are carried out on a Linux 64-bit machine with 32 *GB* of RAM. The results are taken from one testrun, as we haven't found a considerable variance between different runs.

4.1 Platform File Size

As discussed before, the new routing scheme eliminates the need for a full specification of all routing paths through the network. Only the links between two adjacent hosts need to be specified. Platform files were generated using a modified version of the Java `PlatformGenerator` from the "contributed section" of the SimGrid source code repository. The platform generator was used to produce platform files that comply with the Barabási–Albert (BA) network topology model [7]. The generated platform files are based on a given number of clusters and a network topology fulfilling the BA-laws. Each cluster connects its hosts through a gateway host to the rest of the network.

Table 2. SimGrid platform file sizes for an increasing number of clusters in the network

Clusters	Original Size	New Size	% of Original
10	44 K	11 K	25.1 %
100	5.5 M	126 K	2.2 %
200	22.2 M	254 K	1.1 %
300	49.6 M	399 K	0.8 %
400	95.7 M	527 K	0.5 %
800	422.4 M	1.0 M	0.2 %

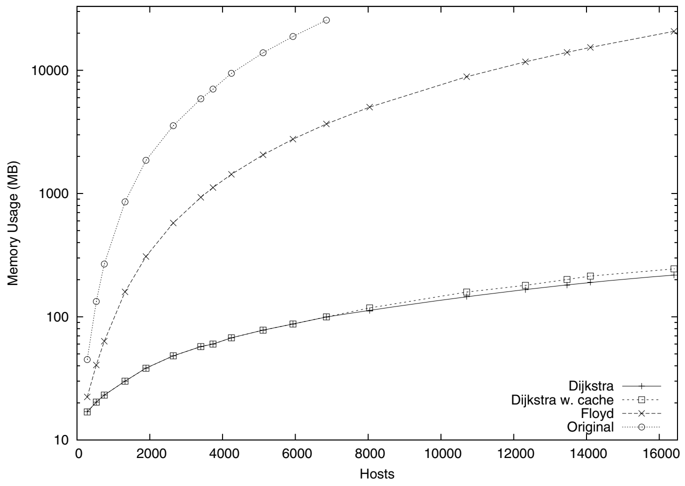


Fig. 2. SimGrid's memory usage in relation to the number of hosts in the network (logarithmic scale)

The second and third column in Table 2 show the platform file sizes respectively for the original and the new platform file structure for different cluster counts. The last column contains the percentage of the new against the original file size. The introduction of a new platform file structure significantly reduces the file size to a fraction of the original size, but it implies losing the ability to describe an arbitrary topology (e.g. a hypergraph).

4.2 Memory Footprint

Memory usage with each of the algorithms is measured by monitoring the `VmSize` value of the running process' status file in the Linux `proc` file system. The `VmSize` is the size of the total address space of a process (not including reserved regions) [8]. The highest value that appeared in this file during a process run is used for the evaluation.

Figure 2 depicts SimGrid's memory usage in relation to the number of hosts[2] specified in the platform file on a logarithmic scale. It is clear that the memory usage of the original SimGrid network representation increases very fast. For example, 20 GB of memory is used at about 6000 hosts. All other algorithms show a significant reduction in memory usage, thus allowing for the simulation of larger networks. The use of Floyd's algorithm significantly reduces the rate of increase and the two variants of Dijkstra's algorithm do so even more. With Floyd's algorithm the simulation requires 20 GB at about 16000 hosts, an improvement of more than 250 %. Furthermore, the Dijkstra algorithms are able to handle up to 16000 hosts with about 200 MB of memory.

4.3 Runtime Performance

To test the runtime performance of the algorithms, the startup time of the simulator and the time it takes to calculate or lookup the routes has to be taken into account. The size of the network affects the startup time for the orginal scheme and Floyd's algorithm, but it has no impact for the Dijkstra schemes. However for the route lookup, the behavior is exactly the opposite. In that case, Dijkstra takes significantly more time to perform the route calculation compared to the other algorithms. Consequently, the amount of simulated traffic as well as the size of the network will influence the performance.

As there is no standard benchmark for the simulator, we focus on the performance of the platform initialization and the route lookups, instead of the duration of a full simulation. We measure the time it takes to complete the initialization functions of SimGrid, namely the `MSG_create_environment` and `MSG_launch_application` functions for the parsing and processing of the platform and deployment files respectively, and the amount of time spent looking up a routing path. We have wrapped the route lookup in a function to be able to profile accurately. This new function is called from the `communicate` function in *SURF*. For the lookup time, we consider the total amount of time spent in the wrapper function divided by the number of calls.

Initialization Performance. Table 3 clearly shows that both variants of Dijkstra's algorithm result in the shortest initialization sequence. The Floyd algorithm requires a significant amount of time to initialize the predecessor and cost matrices. However, storing these on disk can remove this runtime cost for simulations that reuse the same network topology. SimGrid's original routing algorithm is significantly slower in initialization than both Dijkstra variants, particularly for increasing number of hosts.

[2] In the tested development version of SimGrid the number of hosts appeared to be doubled internally compared to the amount of hosts specified by the platform files, resulting in a memory usage that is 4 times as high as it should be for the original and the Floyd route algorithm. This explains the difference between Fig. 2 and the formulas in Table 1.

Table 3. Total time measurement of the platform and deployment file initialization

Clusters	Hosts	Dijkstra	Dijkstra w. cache	Floyd	Original SimGrid
10	280	74 *ms*	73 *ms*	772 *ms*	654 *ms*
50	1319	343 *ms*	353 *ms*	71874 *ms*	16555 *ms*
90	2646	806 *ms*	779 *ms*	608 *s*	68376 *ms*
150	4242	1215 *ms*	1213 *ms*	2578 *s*	183 *s*
180	5112	1483 *ms*	1435 *ms*	4450 *s*	267 *s*
250	6854	1907 *ms*	1968 *ms*	10867 *s*	515 *s*

Table 4. Average duration of a route lookup

Clusters	Hosts	Dijkstra	Dijkstra w. cache	Floyd	Original SimGrid
10	280	7629 μs	367 μs	3 μs	2 μs
50	1319	39403 μs	1672 μs	4 μs	2 μs
90	2646	91058 μs	3854 μs	4 μs	2 μs
150	4242	145 *ms*	6094 μs	5 μs	2 μs
180	5112	173 *ms*	7759 μs	5 μs	2 μs
250	6854	239 *ms*	9828 μs	5 μs	2 μs

Route Lookup Performance. For the route lookup performance, we have measured the execution time of the route calculation function and compared these timings in Table 4. The Floyd algorithm and the original routing algorithm have a negligible route lookup cost, compared to the duration of other calculations in the `communicate` function and the time required for solving SimGrid's analytical network model. Both Dijkstra algorithms perform much worse in this regard. Still, the cache makes a significant difference. The efficiency of the route cache depends on the scenario that is simulated, more specifically on the number of network messages that are sent from the same host. We have measured an increase in average computation time per communication by a factor of 500 for Dijkstra and by a factor of 25 for cached Dijkstra.

5 Future Work

Although our improvements are significant, still more efficient methods can be developed. These may however require more fundamental changes to SimGrid. A more advanced way of representing the network topology in memory can further reduce the memory usage and also improve performance. If SimGrid has a real notion of a clusters and its associated nodes, routing can be carried out between clusters or cluster gateways, partially ignoring the nodes. In a hierarchical network representation, it would also be possible to use different routing algorithms on different levels, each optimized for their specific network topology or application.

6 Conclusion

The simulation of large-scale distributed systems using SimGrid is currently impeded by intensive memory usage of SimGrid's network representation and routing facilities. In addition, the platform files that describe large network configurations require a significant amount of parsing time and disk space. An advantage however, is the low runtime cost of route lookups which is independent of the number of hosts in the network.

In support of simulating large-scale systems with SimGrid, we have presented a number of routing algorithms that considerably reduce the amount of memory required for simulating the network. We have demonstrated that route calculation with Floyd's routing algorithm significantly reduces the memory footprint. Startup costs of the simulation however are heavily affected by the number of hosts, while the route calculation is almost as fast as the original route lookup method. The Dijkstra algorithm results in a major decrease in memory usage, but induces a runtime cost for calculating the routes at each route lookup. In order to mitigate this, a cached version of Dijkstra's algorithm was introduced that induces this cost only the first time a route is asked for, at the cost of slightly higher use of memory. The introduction of these algorithms results in a classical space-time tradeoff. In this regard, we have shown the implications for each algorithm in terms of memory usage and runtime cost.

Acknowledgements

We gratefully thank Arnaud Legrand and Martin Quinson of the SimGrid development team for their constructive comments and inspiring discussions.

References

1. Casanova, H., Legrand, A., Quinson, M.: SimGrid: A generic framework for large-scale distributed experiments. In: Proceedings of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM 2008), Washington, DC, USA, pp. 126–131. IEEE Computer Society, Los Alamitos (2008)
2. Frincu, M.E., Quinson, M., Suter, F.: Handling very large platforms with the new simgrid platform description formalism. Technical Report 0348, INRIA (2008)
3. Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM 5(6), 345 (1962)
4. Dijkstra, E.W.: A note on two problems in connection with graphs. Numerische Mathematik 1, 269–271 (1959)
5. Siek, J., Lee, L.: The boost graph library (BGL) (2000)
6. Austern, M.H.: Generic Programming and the STL: Using and Extending the C++ Standard Template Library. Addison Wesley Longman Inc., Amsterdam (1999)
7. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
8. Bovet, D., Cesati, M.: Understanding the Linux Kernel, 3rd edn. O'Reilly Media, Inc., Sebastopol (2005)

Image Processing and Visualization

Semantic Visual Abstraction for Face Recognition

Yang Cai[1], David Kaufer[2], Emily Hart[3], and Elizabeth Solomon[4]

[1] Ambient Intelligence Lab, [2] English
[3] Electrical and Computer Engineering, [4] Art
Carnegie Mellon University
{ycai,kaufer,erhart,lizzeesolomon}@andrew.cmu.edu
www.cmu.edu/vis

Abstract. In contrast to the one-dimensional structure of natural language, images consist of two- or three-dimensional structures. This contrast in dimensionality causes the mapping between words and images to be a challenging, poorly understood and undertheorized task. In this paper, we present a general theoretical framework for semantic visual abstraction in massive image databases. Our framework applies specifically to facial identification and visual search for such recognition. It accommodates the by now commonplace observation that, through a graph-based visual abstraction, language allows humans to categorize objects and to provide verbal annotations to shapes. Our theoretical framework assumes a hidden layer between facial features and the referencing of expressive words. This hidden layer contains key points of correspondence that can be articulated mathematically, visually or verbally. A semantic visual abstraction network is designed for efficient facial recognition in massive visual datasets. In this paper, we demonstrate how a two-way mapping of words and facial shapes is feasible in facial information retrieval and reconstruction.

Keywords: semantic network, visual abstraction, visual search, human features, face, face recognition, information retrieval, video analytics.

1 Introduction

If a picture is worth 10,000 words [5], can a word be worth 10,000 images? The answer is *yes*. As visual abstractions, many linguistic referring expressions convey visual information with much greater efficiency than visual images. In our everyday life, we detect, recognize and retrieve images with words, which dramatically compress the representational information data space. For example, we often describe a traffic intersection with a letter 'T', or 'X', where we compress an image (e.g. 1 megabyte) to a letter (e.g. 1 byte). We also can retrieve images from our memory with words. This two-way transformation has been culturally supported since the invention of the alphabet. One central objective of this paper is to address how to improve visual search processes by the two-way mapping of images and words. This study will also demonstrate applications to real-world problems, such as the identification of faces

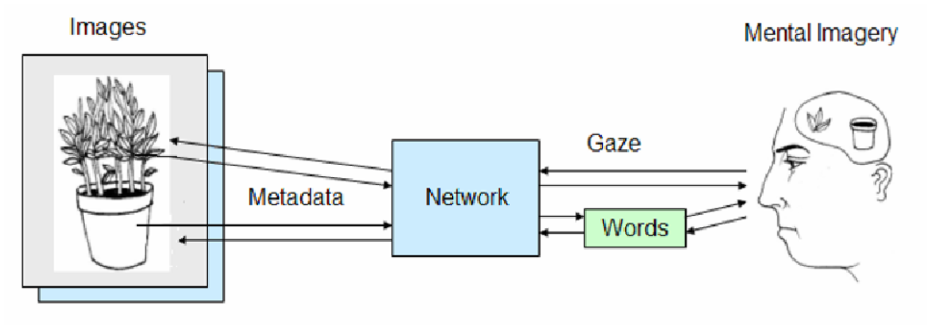


Fig. 1. Overview of the semantic visual abstraction in a network

from a massive video database. Given a method to detect gaze and objects, how do we encode our visual information in multiple resolutions to minimize the communication load and maximize the efficiency for information retrieving? Figure 1 illustrates the architecture of a visual abstraction network.

2 Image-Word Mapping

For many years, cognitive scientists have investigated visual abstraction from psychological experiments including visual search using *foveal vision* [17-27] and *mental rotation* [30]. Visual abstraction models have also been developed, notably Marr's cylinder model of human body structures [28] and the spring-mass graph model of facial structures [29]. Unfortunately, those visual abstractions don't address the image-word two-way mapping issues in particular. Recently, scientists have begun to model the relationship between words and images. CaMeRa [8], for example, is a computational model of multiple representations, including imagery, numbers and words. However, the mapping between the words and images in this system is linear and singular, lacking flexibility. An Artificial Neural Network model has been proposed to understand oil paintings [9], where Solso remarks that the hidden layers of the neural network enable the two-way mapping of words and visual features more effectively. With this method, Solso has argued that fewer neurons are required to represent more images. However, the content of the hidden layers of the neural network remains a mystery.

Because of the two- or three-dimensional structure of images and the one-dimensional structure of language, the mapping between words and images is a challenging and still undertheorized task. Arnheim observed that, through abstraction, language categorizes objects. Yet language, through its richness, further permits humans to create categorizations and associations that extend beyond shape alone [2]. As a rich abstractive layer, language permits categorizations of textures, two- and three-dimensions, and sub-shapes. As an abstractive layer, natural language seems to be the only method we have to satisfactorily describe a human subject. To explore this insight further, Roy developed a computerized system known as Describer that learns to generate contextualized spoken descriptions of objects in visual scenes [10]. Describer illustrates how a description database could be useful when paired with images in constructing a composite image. However, Describer is limited in representing a simplified block world.

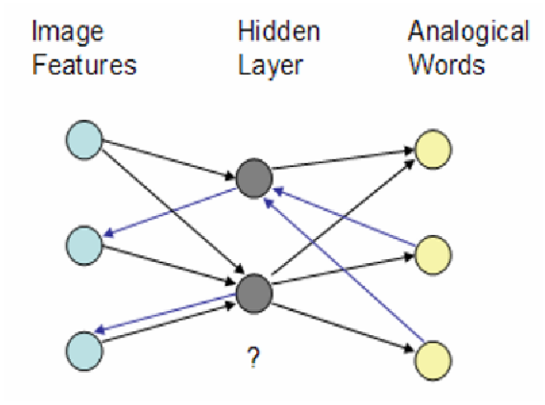


Fig. 2. The two-way mapping neural network model

3 Descriptions for Humans

Our own work and theoretical framework has focused on the mapping between words and images for human features. Why do we focus on human faces? Humans in general and human faces in particular provide among the richest vocabularies of visual imagery in any modern language. Imaginative literature is a well-known source of such descriptions, where human features are often described in detail. In addition, reference collections in the English language focused on visual imagery, such as description and pictorial dictionaries, never fail to have major sections devoted to descriptions of the human face. These sections are typically devoted to anatomical rather than social descriptions of faces based on cultural stereotypes and analogies. The mappings between images and faces we have been exploring are built upon cultural stereotype and analogical associations.

In the following sections, we briefly overview a variety of semantic visual description methods, including multiple resolution, semantic differentiation, symbol-number, and analogy. Then, we introduce the computational implementation of the human description through the interaction of words and images.

4 Multiple Resolution Descriptions

Human descriptions function as classifiers for shape, color, texture, proportion, size and dynamics in *multiple resolutions*. For example, one may start to describe a person's torso, then her hairstyle, face, eyes, nose, and mouth. Human feature descriptions have a common hierarchic structure [1]. For example, figure, head, face, eye, et al. Like a painter, verbal descriptions can be built in multiple resolutions. The words may start with a coarse global description and then 'zoom' into sub components and details. See Fig. 3 for a breakdown of a global description of a human head.

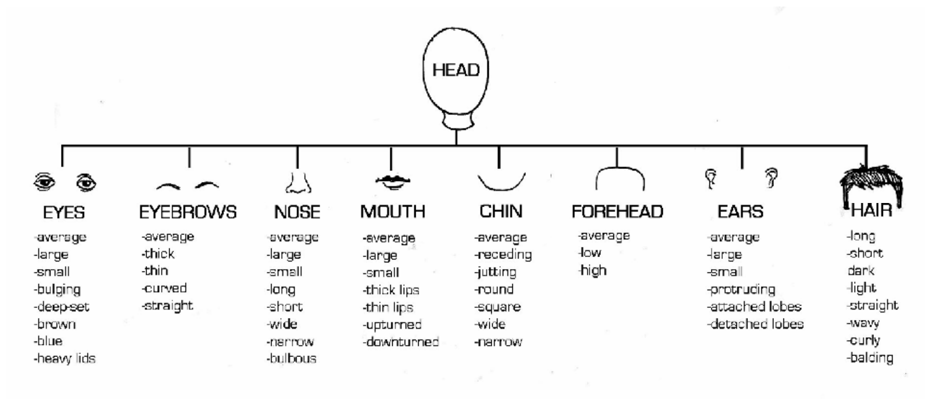


Fig. 3. Multi-resolution representation of a face

In our research to date, we have collected over 100 entries of multi-resolution descriptions from imaginative literature. Our collection ranges from global descriptions to components and details. Due to limitations of space, we will only enlist a few samples, where the underlined sections represent the global levels of description, the **bolded** show the component-based descriptions, and the *italicized* sections, the details:

- "For **A lean face** , pitted and scarred, very **thick black eyebrows** and **carbon-black eyes** with *deep grainy circles of black* under them. A *heavy five o'clock shadow*. But the skin under all was pale and unhealthy- looking. [11]"
- "Otto has a **face** like very ripe peach. **His hair** is fair and thick, growing low on his **forehead**. He has small sparkling **eyes**, full of naughtiness, and a wide, disarming **grin** which is too innocent to be true. When he grins, *two large dimples* appear in his peach **blossom cheeks**.[12]
- "Webb is the oldest man of their regular foursome, fifty and then some- a lean thoughtful gentleman in roofing and siding contracting and supply with a calming gravel voice, his **long face** broken into *longitudinal strips by creases* and **his hazel eyes** almost lost under an amber **tangle of eyebrows**.[13]"

5 Semantic Differential Representation

The Semantic Differential method measures perceptual and cognitive states in numbers or words arrayed on a linear scale. For example, the feeling of pain can be expressed with adjectives, ranging from weakest to strongest. Figure 4 shows a chart of visual, numerical and verbal expressions of pain in hospitals: No Hurt (0), Hurts Little Bit (2), Hurts Little More (4), Hurts Even More (6), Hurts Whole Lot (8) and Hurts Worst (10). This pictorial representations are very useful in patient communication where descriptions of pain type (e.g., pounding, burning) and intensity (e.g., little, a lot) lack a robust differentiated vocabulary.

The physical feeling can be quantified with mathematical models. When the change of stimulus (I) is very small, one won't detect the change. The minimal difference (ΔI) that is just noticeable is called perceptual threshold and it depends on the initial stimulus strength I. At a broad range, the normalized perceptual threshold is a constant, $\Delta I/I = K$. This is Weber's Law [16].

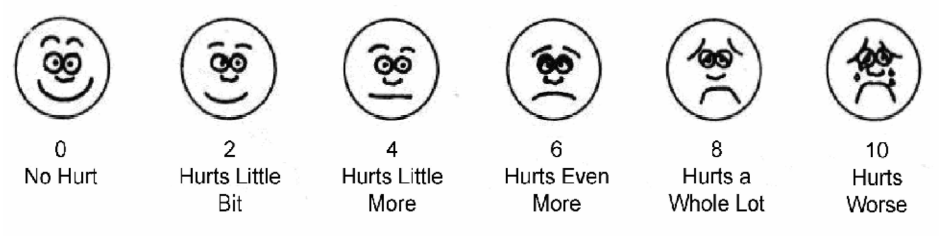


Fig. 4. Expressions of pain in pictures, numbers and words

Given the perceptual strength *E*, as the stimulus *I* changes by ΔI, the change of *E* is ΔE. We have the relationship $\Delta E = K*\Delta I/I$. Let ΔI be *dI* and ΔE be *dE*, thus we have the Weber-Fechner's Law:

$$E = K*ln(I) + C$$

where, *C* is constant and *K* is Weber Ratio, *I* is stimulus strength and *E* is the perceptual strength. Weber-Fechner's Law states that the relationship between our perceptual strength and stimulus strength is a logarithmic function. This perhaps explains why we are able to use limited words to describe a broad range of sensational experiences.

6 Symbol-Number Descriptions

In many cases, numbers can be added to provide even greater granularities. For example, the FBI's Facial Identification Handbook [14] comes with a class name such as bulging

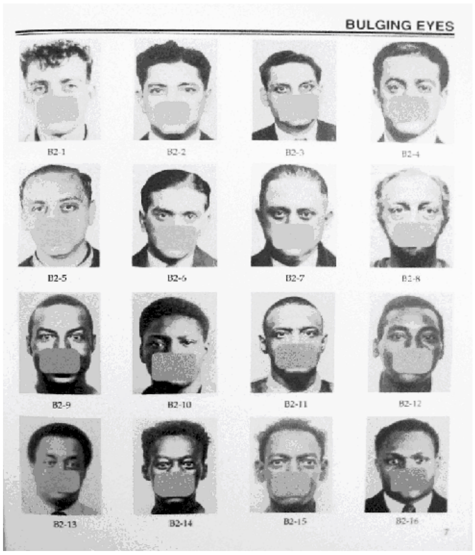


Fig. 5. Bulging Eyes from FBI Facial Identification Catalog

eyes and then a number to give specific levels and types. The FBI has already created a manual for witnesses, victims, and other suspect observers to use in identifying possible suspect features. The catalog presents several images per page under a category such as "bulging eyes." Each image in such a category has bulging eyes as a feature, and the respondent is asked to identify which image has bulging eyes most closely resembling the suspect. See Figure 5 for an example. This book is an extremely efficient and effective tool for both forensic sketch artists and police detectives. It is most commonly used as a tool in helping a witness or victim convey the features of the suspect to the sketch artist in order to render an accurate composite sketch.

7 Analogical Descriptions

From the multi-resolution point of view, an analogy describes in a coarse way in contrast to symbolic-number descriptions. Instead of describing features directly, people often find it more intuitive to refer a feature to a stereotype, for example, a movie star's face. The analogical mapping includes structural mapping (e.g. face to face), or component mapping (e.g. Lincoln's ear and Washington's nose). Children often use familiar objects to describe a person, for example using 'cookie' as an analogical reference for a round face.

Analogies are culture-based. In the Western world, several nose stereotypes are named according to historical figures. Many analogies are from animal noses or plants. Fig. 6 illustrates examples of the nose profiles as described above. We use a simple line drawing to render the visual presentation.

Analogies are a trigger of experience, which involves not only images, but also dynamics. The nose at the far right in Fig. 6 shows a 'volcano nose', which triggers a reader's physical experience such as pain, eruption, and explosion. In this case, readers not only experience it but also predict the consequence. Therefore, it is an analogy of a novel physical process that remains under the visible surface.

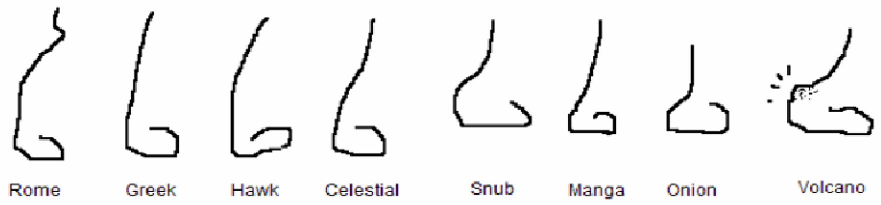


Fig. 6. Analogical description of noses

Given a verbal description of the nose, how do we visually reconstruct the nose profile with minimal elements? In this study, we use a set of 5 to 9 'control points' to draw a profile. By adjusting the relative positions of the control points, we can reconstruct many stereotypes of the profiles and many others in between. To smooth the profile contour, we apply the Spline [15] curve fitting model. See Fig. 7.

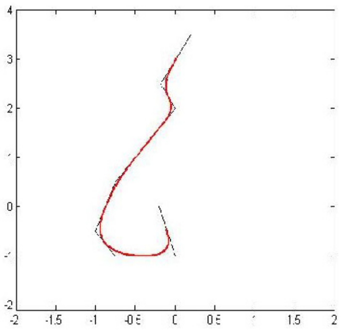


Fig. 7. Reconstructing a nose profile with points (black) and Spline curve (red)

8 The Verbal Description Database for Human Features

In this study, we have also collected over 100 verbal descriptions of human faces from several thesauri and descriptive dictionaries. The structure of the database is as follows: 1) the entity, 2) the side of the body, 3) the region of the body, 4) the part of the body, and 5) subtypes. The database is organized in terms of the resolution based on a hierarchy of human features reduced to each final descriptor. The database is intended to list all possible measurable descriptors of human features including face, body, and movement.

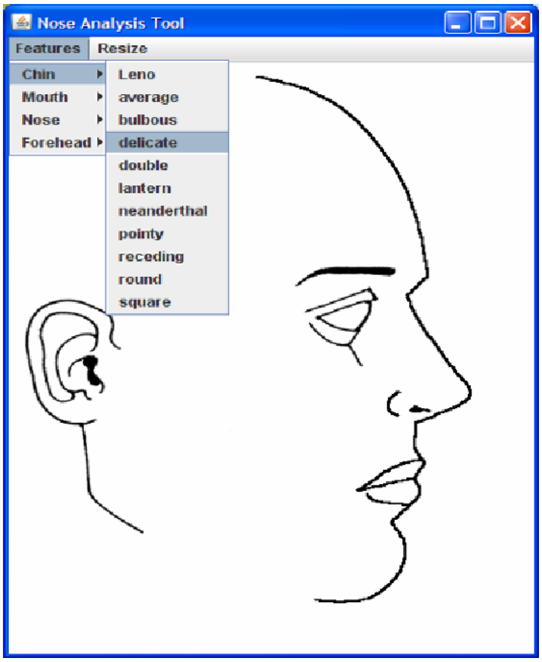
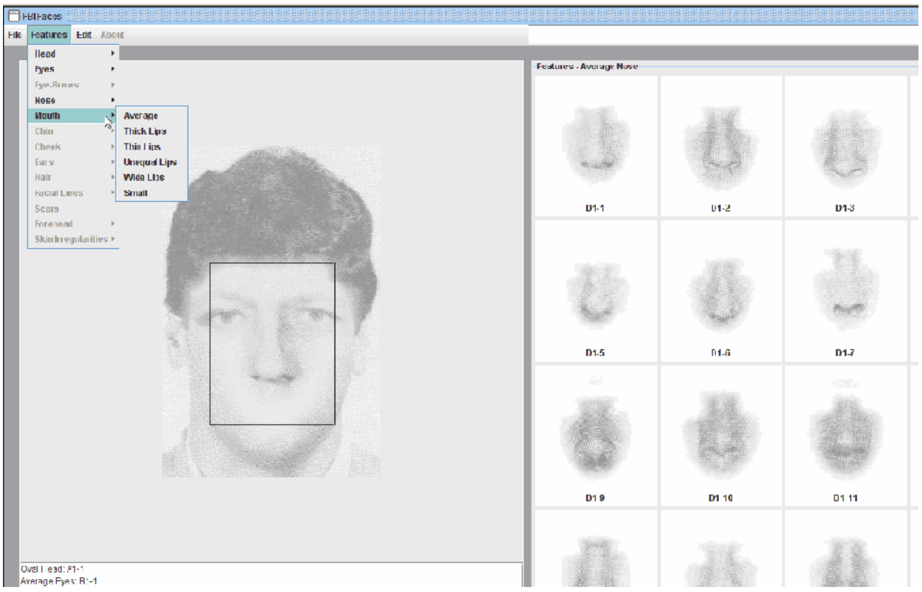


Fig. 8. Interactive facial profile reconstruction based on line-drawing. The code is written in Java so that it is possible to run on the Internet. These descriptions can then be rendered and distributed on the network: http://www.cmu.edu/vis/project9.html

9 Interactive Facial Reconstruction

We developed a computationally working prototype of the interactive system for facial reconstruction. In the system, a user selects the feature keywords in a hierarchical structure. The computer responds to the selected keyword with a pool of candidate features that are coded with labels and numbers. Once a candidate is selected, the computer will superimpose the components together and reconstruct the face. See Fig. 8 and Fig. 9.

As we know, composite sketches of a suspect are typically done by highly-trained professionals. Our system enables inexperienced users to reconstruct a face using only a simple menu driven interaction. In addition, this reconstruction process is reversible. We have designed it for use not only in facial description studies, but also in studies for robotic vision and professional training.



Oval Head: A1-1
Average Eyes: B1-1

Fig. 9. Interactive front facial reconstruction based on image components

10 Conclusions

In this study, we assume a hidden layer between the human perception of facial features and referential words that contain 'control points' that can be articulated mathematically, visually or verbally. Our framework of a semantic network associating verbal and visual information remains in its early stages. Nevertheless, we countenance its long-term promise for understanding how we meaningfully and effortlessly map between visual and verbal information in the successful interpersonal communication about faces. At this moment, we only have profile and frontal facial reconstruction models. In the future, we plan to develop both whole head and body models with far more control points and referencing expressions indexed into those points.

Today, we have an overabundance of data but not nearly enough attention or bandwidth. Image and video collections grow at an explosive rate that exceeds the capacity of network and human attention. In real-time surveillance systems, over a terabyte per hour are transmitted for only a small number of platforms and sensors. We believe that the visual abstraction network described in this paper is one of the feasible solutions that can and should be more thoroughly developed.

Acknowledgement

We would like to thank Army Research Office (ARO) and National Science Foundation (NSF) for their sponsorship. We are also in debt to Brian Zeleznik for his comments and editing.

References

1. Cai, Y.: How Many Pixels Do We Need to See Things? In: Ganter, B., de Moor, A., Lex, W. (eds.) ICCS 2003. LNCS, vol. 2746. Springer, Heidelberg (2003)
2. Arnheim, R.: Visual Thinking. University of California Press (1969)
3. Allport, A.: Visual Attention. MIT Press, Cambridge (1993)
4. Yarbus, A.L.: Eye Movements during Perception of Complex Objects. Plenum Press, New York (1967)
5. Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth 10,000 words. Cognitive Science 11, 65–100 (1987)
6. Geisler, W.S., Perry, J.S.: Real-time foveated multiresolution system for low-bandwidth video communication. In: Proceedings of Human Vision and Electronic Imaging. SPIE, Bellingham (1998)
7. Shell, J.S., Selker, T., Vertegaal, R.: Interacting with groups of computers. Communications of the ACM 46, 40–46 (2003)
8. Tabachneck-Schijf, H.J.M., Leonardo, A.M., Simon, H.A.: CaMeRa: A computational model of multiple representations. Cognitive Science 21, 305–350 (1997)
9. Solso, R.L.: Cognition and the Visual Arts. The MIT Press, Cambridge (1993)
10. Roy, D.: Learning from Sights and Sounds: A Computational Model. Ph.D. In: Media Arts and Sciences, MIT (1999)
11. Doctorow, E.L.: Loon Lake. Random House, New York (1980)
12. Isherwood, C.: Goodbye to Berlin. Signet. (1952)
13. Updike, J.: The Rabbit is Rich. Ballantine Books (1996)
14. FBI Facial Identification Catalog (November 1988)
15. Spline (2007), `http://en.wikipedia.org/wiki/Spline_(mathematics)`
16. Li, Q., Rosa, M.D., Daniela, R.: Distributed Algorithms for Guiding Navigation across a Sensor Network. Dartmouth Department of Computer Science (2003)
17. Wolfe, J.M.: Visual Search. In: Pashler, H. (ed.) Attention, East Sussex. Psychology Press, UK (1998)
18. Theeuwes, J.: Perceptual selectivity for color and form. Perception & Psychophysics 51, 599–606 (1992)

19. Treisman, A., Gelade, G.: A feature integration theory of attention. Cognitive Psychology 12, 97–136 (1980)
20. Verghese, P.: Visual search and attention: A signal detection theory approach. Neuron 31(13), 523–535 (2001)
21. Visual Seach (2008), `http://en.wikipedia.org/wiki/Visual_search`
22. Yarbus, A.L.: Eye Movements during Perception of Complex Objects. Plenum Press, New York (1967)
23. Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth 10,000 words. Cognitive Science 11, 65–100 (1987)
24. Duchowski, A.T., et al.: Gaze-Contingent Displays: A Review. Cyber-Psychology and Behavior 7(6) (2004)
25. Kortum, P., Geisler, W.: Implementation of a foveated image coding system for image bandwidth reduction. In: SPIE Proceedings, vol. 2657, pp. 350–360 (1996)
26. Geisler, W.S., Perry, J.S.: Real-time foveated multiresolution system for low-bandwidth video communication. In: Proceedings of Human Vision and Electronic Imaging. SPIE, Bellingham (1998)
27. Majaranta, P., Raiha, K.J.: Twenty years of eye typing: systems and design issues. In: Eye Tracking Research and Applications (ETRA) Symposium. ACM Press, New Orleans (2002)
28. Marr, D.: Vision. W.H. Freeman, New York (1982)
29. Ballard, D.H., Brown, C.M.: Computer Vision. Prentice-Hall Inc., New Jersey (1982)
30. Mental Rotation (2007), `http://en.wikipedia.org/wiki/Mental_rotation`

High Frequency Assessment from Multiresolution Analysis

Tássio Knop de Castro, Eder de Almeida Perez, Virgínia Fernandes Mota, Alexandre Chapiro, Marcelo Bernardes Vieira, and Wilhelm Passarella Freire

Universidade Federal de Juiz de Fora, DCC/ICE,
Cidade Universitária, CEP: 36036-330, Juiz de Fora, MG, Brazil
{tassioknop,eder.perez,virginia.fernandes,alexandre.chapiro,
marcelo.bernardes,wilhelm.freire}@ice.ufjf.br
http://www.gcg.ufjf.br

Abstract. We propose a method for the assessment and visualization of high frequency regions of a multiresolution image. We combine both orientation tensor and multiresolution analysis to give a scalar descriptor of high frequency regions. High values of this scalar space indicate regions having coincident detail vectors in multiple scales of a wavelet decomposition. This is useful for finding edges, textures, collinear structures and salient regions for computer vision methods. The image is decomposed into several scales using the Discrete Wavelet Transform (DWT). The resulting detail spaces form vectors indicating intensity variations which are combined using orientation tensors. A high frequency scalar descriptor is then obtained from the resulting tensor for each original image pixel. Our results show that this descriptor indicates areas having relevant intensity variation in multiple scales.

Keywords: high frequency detection, multiresolution analysis, orientation tensor.

1 Introduction

The evaluation of high frequencies in an image is an important task for several applications in computer vision, computer graphics and image processing. Objects in a scene are mainly distinguished by the contrast of their borders against a background. In a signal processing point of view, this can be seen as brightness variation with multiple frequencies.

However, object and background areas can be arbitrarily complex. One way of estimating salient regions is to use multiresolution to capture global and local brightness variations. Even in a non-redundant wavelet decomposition, local and global borders occurring in the same region may carry useful information. The problem is to combine the global information into a single image. In this sense, orientation tensors can capture the multivariate information of several scales and color channels [1].

In this paper, we combine both orientation tensor and multiresolution analysis to give a scalar descriptor of high frequency regions. High values of this scalar space indicate regions having coincident detail vectors in multiple scales of wavelet decomposition. This is useful for finding edges, textures, collinear structures and salient regions for computer vision methods.

2 Related Work

Orientation tensors can be used to analyze and draw conclusions about the quality of an image. In Fronthaler et al. [2], the objective is to distinguish noisy content from possible non-trivial structures in biometric assessments. The orientation tensor is decomposed into symmetric representations from which a particular definition of quality can be estimated.

Wong and Chung [3] use orientation tensors to exploit local structural coherence to improve the quality of the binary segmentation of an image. An estimation of the local structural orientation through eigen decomposition of these tensors is performed for local structure description.

In Han and Shi [4], the wavelet transform plays an important role in the task of decomposing a texture image into several levels. Once a decomposition level is chosen, textures are then removed from the original image by the reconstruction of low frequencies only.

Bigun et al. [5] use a structure tensor to represent and detect more intricate patterns than straight lines and edges to produce and filter dense orientation fields for feature extraction, matching, and pattern recognition.

Schou et al. [6] propose a method to detect line and edge structures in multichannel remote sensing images. They also use tensors to represent orientation information. Vliet and Faas [7] decompose structure tensors to analyze and represent multiple oriented structures inside a local neighborhood of an image. They propose cluster analysis to divide the local gradient vectors that would normally construct a single tensor into a limited number of clusters.

Most of the related works use orientation tensors or multiresolution as a step to gather specific image information in a single scale. In this paper, a weighted sum of orientation tensors, obtained from several multiresolution scales, is used to combine high frequencies in only one tensor field. This resulting tensor field captures regions having coincident high frequencies that can be used to detect salient areas.

3 Fundamentals

3.1 Wavelets

The wavelet transform decomposes signals over dilated and translated wavelets [8]. A wavelet is a function $\psi \in L^2(\Re)$ with a zero average:

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0 \tag{1}$$

It is normalized $||\psi|| = 1$, and centered in the neighborhood of $t = 0$. A family of time-frequency atoms is obtained by scaling ψ by s and translating it by u:

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right) \tag{2}$$

We are interested in wavelets which form a base of $L^2(\Re^2)$ to represent images. If we have an orthonormal wavelet basis in $L^2(\Re)$ given by ψ with the scaling function ϕ, we can use

$$\psi^1(x) = \phi(x_1)\psi(x_2),\ \psi^2(x) = \psi(x_1)\phi(x_2),\ \psi^3(x) = \psi(x_1)\psi(x_2) \tag{3}$$

to form an orthonormal basis in $L^2(\Re^2)$ [8].

$$\{\psi^1_{j,p}, \psi^2_{j,p}, \psi^3_{j,p}\}_{[j,p]\in Z^3} \tag{4}$$

In this paper, we define a vector $v_{j,p} \in \Re^3$ given by the inner product

$$v_{j,p} = [I \cdot \psi^1_{j,p}, I \cdot \psi^2_{j,p}, I \cdot \psi^3_{j,p}]^T \tag{5}$$

at scale j and position $p \in I$, where I is the input image.

3.2 Orientation Tensor

A local orientation tensor is a special case of non-negative symmetric rank 2 tensor, built based on information gathered from an image. As shown by Knutsson [1], such a tensor can be produced by combining outputs from polar separable quadrature filters. Because of its construction, such a tensor has special properties and contains valuable information about said image.

From the definition given by Westin [9], orientation tensors are symmetric, and thus an orientation tensor T can be decomposed using the Spectral Theorem as shown in (6), where λ_i are the eigenvalues of T.

$$T = \sum_{i=1}^{n} \lambda_i T_i \tag{6}$$

If T_i projects onto a m-dimensional eigenspace, we may decompose it as

$$T_i = \sum_{s=1}^{m} e_s e_s^T \tag{7}$$

where $\{e_1,...,e_m\}$ is a base of $\Re^m$. An interesting decomposition of the orientation tensor T proposed by Westin [9] is given by

$$T = \lambda_n T_n + \sum_{i=1}^{n-1} (\lambda_i - \lambda_{i+1}) T_i \tag{8}$$

where λ_i are the eigenvalues corresponding to each eigenvector e_i. This is an interesting decomposition because of its geometric interpretation. In fact, in $\Re^3$, an orientation tensor T decomposed using (8) can be represented by a spear (its main orientation), a plate and a ball

$$T = (\lambda_1 - \lambda_2)T_1 + (\lambda_2 - \lambda_3)T_2 + \lambda_3 T_3. \tag{9}$$

A $\Re^3$ tensor decomposed by (9), with eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$, can be interpreted as following:

- $\lambda_1 >> \lambda_2 \approx \lambda_3$ corresponds to an approximately linear tensor, with the spear component being dominant.
- $\lambda_1 \approx \lambda_2 >> \lambda_3$ corresponds to an approximately planar tensor, with the plate component being dominant.
- $\lambda_1 \approx \lambda_2 \approx \lambda_3$ corresponds to an approximately isotropic tensor, with the ball component being dominant, and no main orientation present.

Consider two orientation tensors A and B and its summation $T = A + B$. After the decomposition of T using (9), the component $(\lambda_1 - \lambda_2)T_1$ is an estimate of the collinearity of the main eigenvectors of A and B.

4 Proposed Method

The method proposed in this paper uses high frequency information extracted from wavelet analysis. For each scale j, we create a vector based on (5). This vector contains the high frequency value at vertical, horizontal and diagonal directions of the image I at the position p and scale j. Symmetric rank 2 tensors are then created as

$$M_{j,p} = v_{j,p} v_{j,p}^T. \tag{10}$$

We find the final tensor $M_{0,p}$ for each pixel of the original image using

$$M_{0,p} = \sum_{j=1}^{n_j} k_j M_{j,p} \tag{11}$$

to combine the tensors obtained at each scale j, where n_j is the number of scales and $k_j \in \Re$ is the weight assigned to each scale, given by

$$k_j = \frac{\sum_{n=1}^{n_p} \text{Trace}(M_{j,n})}{\sum_{k=1}^{n_j} \sum_{n=1}^{n_p} \text{Trace}(M_{k,n})}, \tag{12}$$

where n_p is the number of pixels and Trace$(M_{j,p})$ is the sum of the eigenvalues of $M_{j,p}$. The trace represents the amplification driven by the tensor to the unit sphere and is a good estimator of its importance. Thus, the tensor sum is weighted by the proportion of energy of each scale in the multiresolution pyramid.

In order to find $M_{j,p}$ in (11), we use bilinear interpolation of the tensor values, relative to each position p in the initial image, at the subsampled image at scale j to find the resulting tensor $M_{j,p}$ for each pixel of the initial image. This is

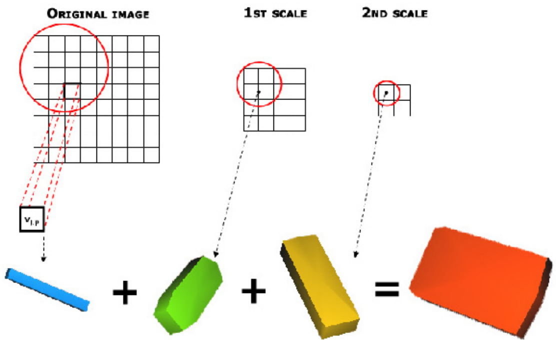


Fig. 1. A tensor is computed for each pixel in original image by a weighted sum of corresponding tensors in each scale. In this example, two wavelet decompositions are performed.

depicted in Fig. 1, where tensors are represented as superquadric glyphs whose longer axis shows the main direction.

Note that the tensor presented in (11) is a 3×3 positive symmetric matrix with real coefficients, and thus we may apply (9). We then find the main orientation component (spear) of the final orientation tensor for each pixel of the input image. This component indicates the collinearity of the interpolated tensors and provides interesting results.

4.1 Implementation

The proposed algorithm consists of three main steps: a discrete wavelet transform [8,10], a tensor field computation and a weighted sum of the computed tensors. The whole process is illustrated in Fig. 2.

The number of scales to be used is a parameter of the algorithm. The DWT splits the image into three detail components and one scale component in the beginning of each iteration. In the next iteration, the same process is applied, using the resulting scale component as the input image.

For each pixel of the input image, its correspondent position at the current scale is computed with subpixel precision for each resolution. The four nearest pixels in a given resolution are used to compute the final tensor. The vectors $v_{j,p}$ described in (5) are computed for each of these pixels and then used to compute four spear type tensors. The final tensor for the subpixel position is obtained by combining these four tensors with bilinear interpolation. The pixel tensor is computed by combining the n_j tensors as showed in (11).

The pixel tensors are decomposed and their eigenvalues are then extracted. The values λ_1 - λ_2 are computed and normalized to form the output image. Color images are split into three monochromatic channels (Red, Green and Blue) and the proposed algorithm is applied to each channel separately. The tensors for each color channel are summed before eigen decomposition.

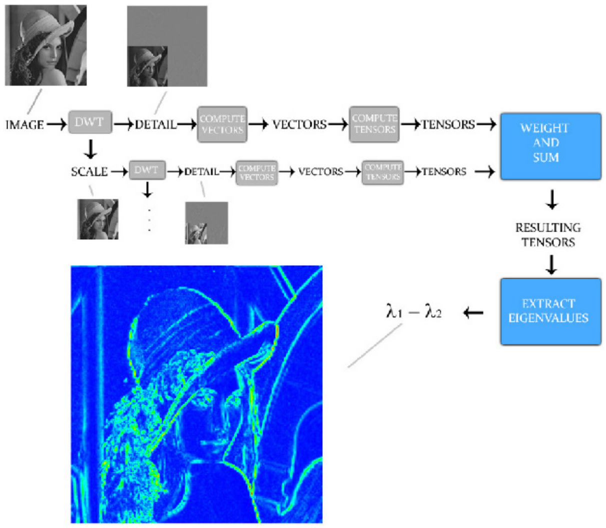


Fig. 2. Example of the proposed algorithm using Daubechies1 to decompose the image into two scales

The complexity of the whole process is $O(n_j \cdot n_p)$, where n_j is the number of analyzed scales and n_p the amount of input pixels. Thus, this is an efficient method that can be further parallelized.

5 Experimental Results

The first experiment consists of fixing an input image and varying the wavelet function and the amount of analyzed scales. This is shown in Figures 3 and 4 where the DWT is applied with different analyzing Daubechies filters and number of scales.

Comparing the Fig. 3b with the Fig. 3c, one may see that the number of scales is important to capture the coarse detail from the image. Note that the church's floor has low frequencies that cannot be detected using only one scale Fig. 3b. This is even clearer in Figures 4b and 4c. The ceiling is formed by coincident frequencies on its geometric details. These details can be better observed in Fig. 4c.

Changing the analyzing filter from Daubechies1 to Daubechies3 provides a better estimation of soft edge transitions. Figures 3b, 3d, 4b and 4d illustrate this behavior.

The resulting eigenvectors associated to the greatest eigenvalues λ_1 are shown in Fig. 5. The tensors eigenvectors are overlayed with the original image using the thermal color to indicate $\lambda_1 - \lambda_2$. Note that they indicate regular patterns in high frequency regions.

In general, it can be noted that high frequencies occurring in the same region at different scales are highlighted by this method. The thermal coloring is a

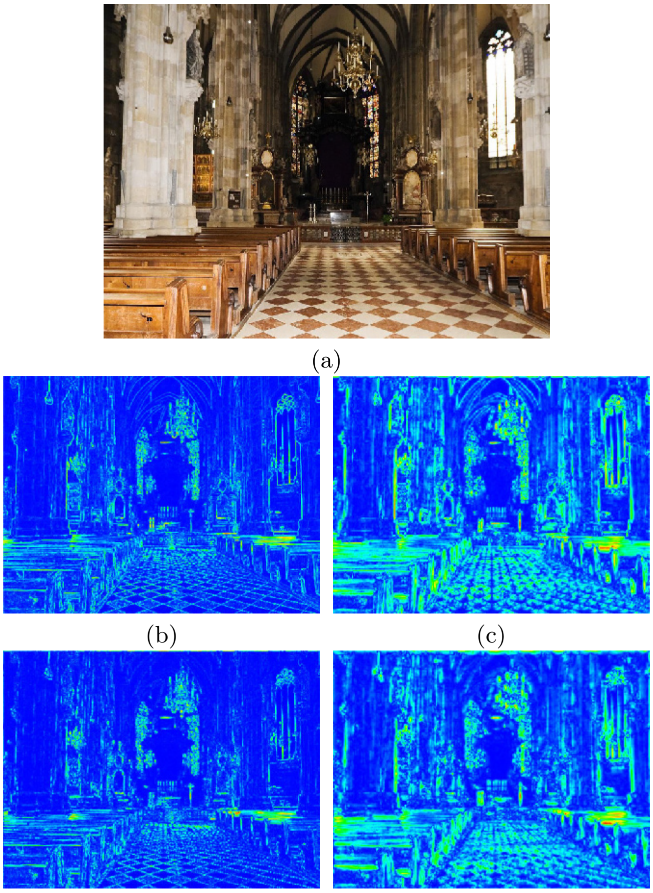


Fig. 3. (a) input image. (b) λ_1 - λ_2 with Daubechies1 and 1 scale. c) Daubechies1 and 3 scales. d) Daubechies3 and 1 scale. e) Daubechies3 and 3 scales.

smooth transition from blue to red, where blue means absence of high frequencies, and red means presence of high frequencies. The green regions also indicate high frequencies, but not as intense as those indicated by red regions. The tensors obtained in the red regions have better estimation of higher frequencies.

The second experiment shows the time spent to apply the algorithm in color images. The Fig. 6 shows the time in seconds in function of the number of scales and image size. One may see the linear behavior of the algorithm, where the slope is the number of scales. However, it is important to note that the algorithm response time may be a bottleneck in real time applications if the number of pixels is high. All experiments were performed on an Intel Core2 Duo 1.8Ghz CPU using a 32bit compiler.

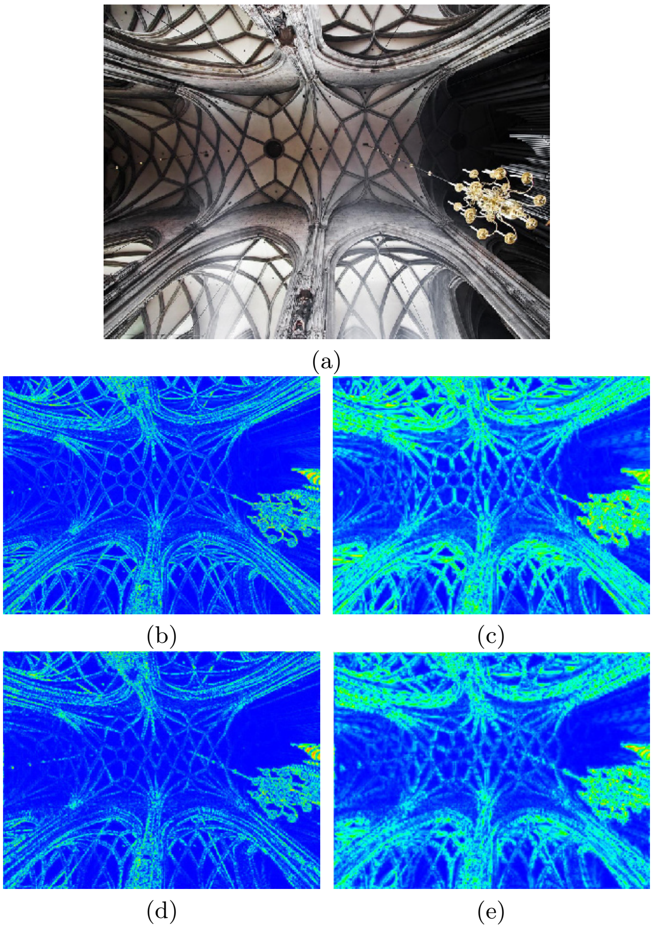


Fig. 4. (a) input image. (b) λ_1 - λ_2 with Daubechies1 and 1 scale. c) Daubechies1 and 3 scales. d) Daubechies3 and 1 scale. e) Daubechies3 and 3 scales.

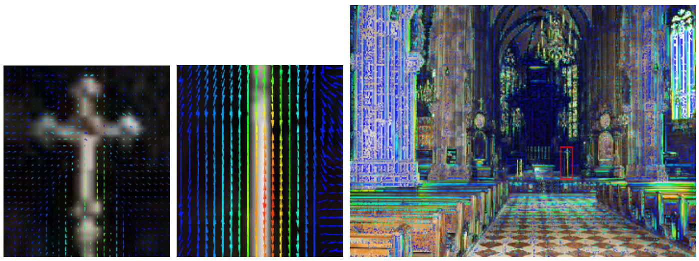


Fig. 5. Eigenvector field overlayed with the input image

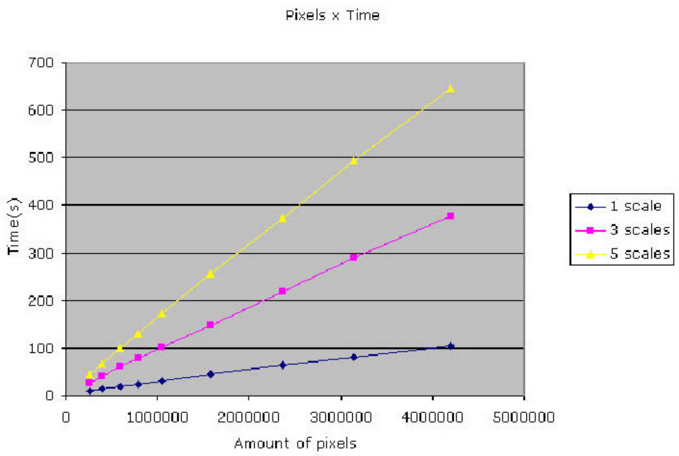


Fig. 6. Evaluation of the running time in function of the number of scales and amount of pixels of a color image

6 Conclusions and Future Works

A method for high frequency assessment and visualization was proposed. It is based on the DWT decomposition and detail information merging using orientation tensors. This multiresolution analysis showed to be suitable for detecting relevant edges and salient areas in an image. Due to the multivariate nature of tensors, the process can be easily applied in color images.

The experimental results show that the high frequency information can be inferred by varying the DWT filters and number of scales. Coincident frequencies in space domain are successfully highlighted. By tuning the number of scales, one may infer texture feature regions. As shown, the linear complexity is suitable for high performance processes.

The $\lambda_1 - \lambda_2$ scalar field is one of the most used orientation alignment descriptors. However, other relations can be extracted from final pixel tensors. Future works should evaluate this remaining tensor information. As an example, there is promising information coded in the tensor eigenvectors. It is also interesting to investigate the tensor field instead of isolated tensors.

The discrete wavelet transform and the tensor summation can be easily parallelized. The use of rising technologies like GPGPUs and multicore CPUs turns this method attractive for high performance applications.

Acknowledgments

Authors thank *Fundação de Amparo à Pesquisa do Estado de Minas Gerais* /FAPEMIG and PROPESQ/UFJF for the financial support of this research.

References

1. Knutsson, H.: Representing local structure using tensors. In: The 6th Scandinavian Conference on Image Analysis, Oulu, Finland, 244–251 Report LiTH–ISY–I–1019, Computer Vision Laboratory, Linköping University, Sweden (June 1989)
2. Fronthaler, H., Kollreider, K., Bigun, J.: Automatic image quality assessment with application in biometrics. In: CVPRW 2006: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop, Washington, DC, USA, p. 30. IEEE Computer Society Press, Los Alamitos (2006)
3. Wong, W.C.K., Chung, A.C.S.: Bayesian image segmentation using local iso-intensity structural orientation. IEEE Transactions on Image Processing 14(10), 1512–1523 (2005)
4. Han, Y., Shi, P.: An adaptive level-selecting wavelet transform for texture defect detection. Image Vision Comput. 25(8), 1239–1248 (2007)
5. Bigun, T.: Recognition by symmetry derivatives and the generalized structure tensor. IEEE Trans. Pattern Anal. Mach. Intell. 26(12), 1590–1605 (2004); Fellow-Josef Bigun and Student Member-Kenneth Nilsson
6. Schou, J., Dierking, W., Skriver, H.: Tensor based structure estimation in multi-channel images. In: Geoscience and Remote Sensing Symposium, Proceedings. IGARSS, IEEE 2000 International, vol. 2, pp. 663–665 (2000)
7. van Vliet, L.J., Faas, F.G.A.: Multi-orientation analysis by decomposing the structure tensor and clustering. In: ICPR 2006: Proceedings of the 18th International Conference on Pattern Recognition, Washington, DC, USA, pp. 856–860. IEEE Computer Society Press, Los Alamitos (2006)
8. Mallat, S.: A Wavelet Tour of Signal Processing (Wavelet Analysis & Its Applications), 2nd edn. Academic Press, London (1999)
9. Westin, C.F.: A Tensor Framework for Multidimensional Signal Processing. PhD thesis, Department of Electrical Engineering Linköping University (1994)
10. Barnard, H.J.: Image and Video Coding Using a Wavelet Decomposition. PhD thesis, Delft University of Technology, Department of Electrical Engineering, Information Theory Group, P.O.Box 5031, 2600 GA, Delft (1994)

Virtual Human Imaging

Yang Cai[1], Iryna Pavlyshak[1], Li Ye[1], Ryan Magargle[2], and James Hoburg[1]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
ycai@cmu.edu
[2] Ansoft, Inc., Pittsburgh, PA, USA
rmagargle@ansoft.com

Abstract. Given 3D scanned anthropological models and the physical parameters of a microwave imaging system, we develop a virtual human surface imagery system with a finite multi-physics surface model. The concealed object detection algorithms are developed based on the wave intensity and surface characteristics. The virtual human image system can be integrated into a systematic design process, enabling multidisciplinary innovations in security, privacy, healthcare, computer vision, and information visualization. This forward-thinking approach intends to transform the development of human imaging technologies from being device-specific and proprietary to being device-independent and open source-oriented. It also transforms the research into a systematic design process, enabling multidisciplinary innovations in digital human modeling, computer vision, information visualization, and computational aesthetics. This study can help to design privacy-aware imaging systems in airports and medical systems.

Keywords: human body, feature recognition, 3D scan, security, privacy.

1 Introduction

The goal of this study is to build a general computational model for designing and evaluating human imaging technologies before a physical system is built. This forward-thinking approach intends to transform the development of human imaging technologies from being device-specific and proprietary to being device-independent and open source. This also transforms imaging research into a systematic design process, which requires multidisciplinary innovations in digital human modeling, computer vision and information visualization.

For example, the growing demand for three-dimensional holographic imaging systems has created significant interests in many disciplines. Current devices operate using a millimeter wave transceiver to reflect the signal off the human body and any objects carried on it. These devices penetrate items that are less dense, such as clothing and hair [5,6,7,8,9,10,11,12,13,14,15,45]. Unlike the latest metal detectors, the system can also detect non-metal threats or contraband, including plastics, liquids, drugs and ceramic weapons hidden under clothing. These high-resolution scanned images reveal intimate bodily details and have raised serious privacy concerns.

Most of the research and development of human scanning systems has been done through unpopular projects in a few private companies or in government laboratories.

As a result, most of the technologies are either device specific or proprietary, which has slowed down the overall advancement of privacy technologies for the 3D body scanning systems.

The following problems warrant a scientific investigation: 1) Given the available databases of anthropological models and the physical parameters of human imaging systems, we simulate the scanning imagery data to be used as an open source for broader research communities; 2) We develop effective algorithms to find the human surface features from the 3D scanning data; Finally, 3) we develop the algorithms to discern concealed objects from the human body. Fig. 1 shows an illustration of the framework.

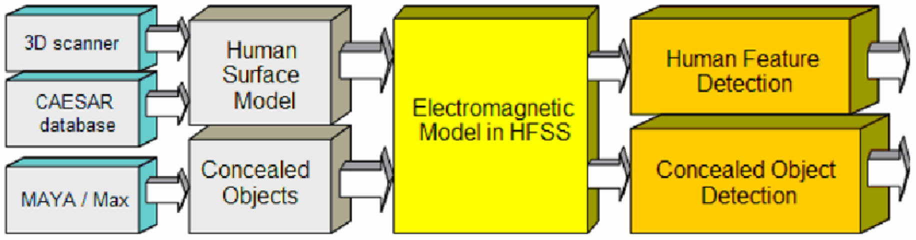


Fig. 1. The framework of the multidisciplinary modeling process that merges at least four domains: computer simulation, computer vision, information visualization and human-centered computing

The *physically augmented virtual human model* is the central idea in the study. In the world of medical research and development, scientists often use so-called ‘phantoms’ to calibrate a new medical instrument. Affordable phantom databases and artifacts, such as Mathworks’ MRI brain phantom images [33], National Library of Medicine’s Visible Humans [34] and DARPA’s Digital Soldier [35], significantly reduce development cycles and increase opportunities for interdisciplinary collaboration and education. Currently, there is no shared scientific benchmarking database in the security human scanning area. In this project, we will develop digital human models that not only contain finite surface elements but also physical properties, for example the reflection of microwave beams on the skin and concealed objects beneath clothing. This requires high-fidelity modeling within a high frequency (900 MHz to 33 GHz) electromagnetic field simulation, which reaches the limit of current physical computation technologies. Compared to MRI imaging simulation, this task is more computationally challenging. We envision that our results will inspire a new area of virtual imaging technologies. The difficulties of our proposed project include: mapping the imperfect laser scanning surface data to the finite element material data, formulating the electromagnetic exciting sources, and calibrating the simulated model.

The *algorithm for detecting human surface features* enables us to segment the human body and reduce the search space for anomalous objects. Many machine learning algorithms are coordinate-dependent and limited by the training data space, for example, artificial neural networks [44]. Some algorithms only work within small bounding boxes that do not deliver an acceptable performance. For example, if a feature detection algorithm takes one hour to process, then it is not useful for a security screening system [31,32]. In this project, we want to develop a model that is invariant to poses and coordinates. From a computer vision point of view, detecting features from 3D body scan data is nontrivial because human bodies are diverse. The technical methodology of function fitting has been used for extracting special landmarks, such as ankle joints, from 3D body scan data [31,32], similar to the method for extracting landmarks on terrain [21,22]. Curvature calculation is also introduced from other fields such as the sequence dependent curvature structure of DNA [19,20]. These curvature calculations use methods such as chain code [30], circle fit, ratio of end-to-end distance to contour length, ratio of moments of inertia, and cumulative and successive bending angles. Curvature values are calculated from the data by fitting a quadratic surface over a square window and then calculating the directional derivatives of this surface. Sensitivity to data noise is a major problem in both the function fitting and curvature calculation methods because typical 3D scanning data is very noisy. Template matching appears to be a promising method because it is invariant to the coordinate system [31,32]. However, defining a template and where to match the template is challenging because it is unique to each particular feature.

How to develop a *discriminative algorithm* to distinguish anomalous objects from human parts is a challenge. In this study, we focus on surface and density characteristics, where the objects can be clustered and highlighted based on the spatial curvature and spatial density of object data points. Artificial anomalous objects embedded into the realistic 3D datasets are used to evaluate the performance of the developed algorithms. This task is the most difficult but most important in the project.

2 Physically Augmented Virtual Human Model

We have developed a set of full-scale virtual human models based on the digital surface scanning data from CAESAR database (with necessary license agreement), which contains 50 males and 50 females aged 16-65, where 50 of them are North American, 24 are Asian, and 26 are from the European survey of Italy, the Netherlands and other countries. As we know, all models in the database have feature landmarks which are important anthropomorphic measurements. We keep them in our test-bed. However, all models wore tight underwear. Therefore, we have to remove that by applying a low-pass filter.

In addition, we also use the state-of-art high fidelity laser 3D scanner[1] to collect our own 20 samples as references. For these in-house models, we will manually annotate the human landmarks. Fig. 2 shows a sample of a 3D human body scanning data and the output of the microwave imaging simulation from HFSS[2].

We input the human scan model to High Frequency Simulation System (HFSS) where we assign the microwave reflection properties to the 3D surface point clouds. This is a non-trivial task because no one has done the full-body microwave imaging simulation with HFSS before. According to state-of-the-art microwave simulation technology, the approach is feasible but very challenging due to the limitations of the

[1] www.creaform3d.com

[2] http://www.ansoft.com/products/hf/hfss/

current software and computers. If we succeed, it would bring a brand new direction for virtual prototyping of imagery devices. To accomplish the task within the limited time and budget, we simplify the computing problems. For example, to reduce the mesh resolution to fit the capacity of the HFSS model, we use the snapshots of frequencies instead of frequency sweeping process and model only portions of the body instead of the whole.

We researched the related patents in order to reverse engineer the technical details. We have found at least two kinds of systems, for example, the Ka-band (27 – 33 GHz) device and Ku band (12 – 18 GHz) device. Due to the different wavelengths and scanning methods, the returning signals are different. In addition, we add anomalous objects such as concealed weapons like guns and knives. Fig. 2 shows an example of the data with the artificial noises and anomalous artifacts in voxels.

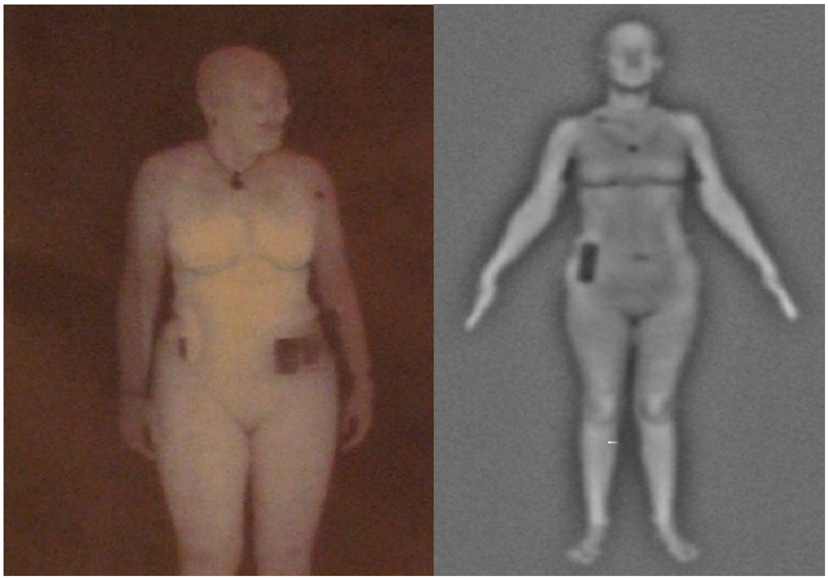


Fig. 2. Real scan image from London Airport (left) and the synthetic image (right)

3 Algorithm for Detecting Anomalous Objects on Skin

Effectively detecting anomalous objects and distinguishing them from human body is the ultimate purpose of the system. As a result, it can significantly suppress the human body details as a background. Removing all the human background information may be not desirable because we need the location and size references. There are many anomaly detection methods: bump-hunting, voxel intensity based, curvature-based and spatial density based clustering, and so on [36-43]. In this project, we develop two spatial analysis models for the anomaly detection: an intensity based detection model and a curvature-based model based detection model.

3.1 Intensity-Based Detection

Metal weapons have their own signatures of intensity properties. However, there are too many kinds of non-metal weapons, such as liquid explosives, which emit different intensity signals. A fixed threshold won't work. In this study, we use HFSS to simulate the scattered radio waves from the objects and human body.

A typical human scanner's wave range is between 500 MHz to 33 GHz, which is a great challenge to simulate the whole body imaging at the resolution of 1 mm with the existing computing resources. To simplify the problem, we crop the 3D human model to a solid 1 x 1 x 0.5 ft^3 slab with a metal gun on the skin. We use the material property for the body with a permittivity and conductivity matching that of sea water (epsilon_r = 81, and conductivity = 4 S/m). We have a material for human muscle, but it is only valid up to 6GHz (epsilon_r ~= 50, conductivity ~= 6S/m), so we chose to stick with sea water and wouldn't expect a significant difference for the qualitative purposes of this simulation. The gun has the properties of copper. Here is a result for the scattered electric field due to a 1V/m incident plane wave:

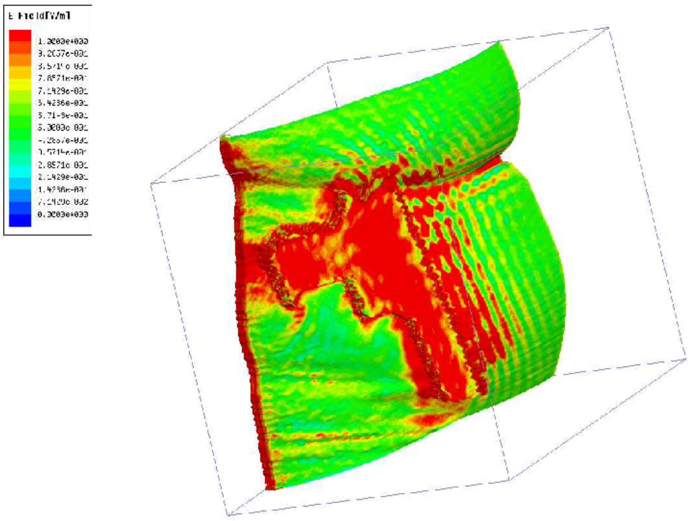


Fig. 3. HFSS simulation of the wave intensive image of human body with a gun at 6 GHz

This is the plot of the magnitude of the electric field at a fixed phase. If we plotted for a sequence of phases between 0 and 180, we would see the field magnitude propagate across the body. Note that the area occupied by the gun has magnitude values near the full 1 V/m, indicating that it reflects most of the signal, whereas the body reflects only around half of that. If we look closely on the left side of the body where it intersects the bounding box, we can see a region of high field value (red). This is due to an artificial resonance with the gun and the boundary condition on the box. For qualitative purposes, the effect on the rest of the simulation should localized and negligible.

Based on the simulated image samples of known materials, we can train an artificial neural network (e.g. Radial Basis Function) to recognize the signature intensity [30]. To adapt to a broader range of data and a greater noise level, we will preprocess the data with normalization algorithms and filters.

3.2 Surface Based Detection

In the continuous case, curvature is defined as the rate of change of slope. In our case, the discrete space, the curvature description must be slightly modified to overcome difficulties resulting from violation of curve smoothness.

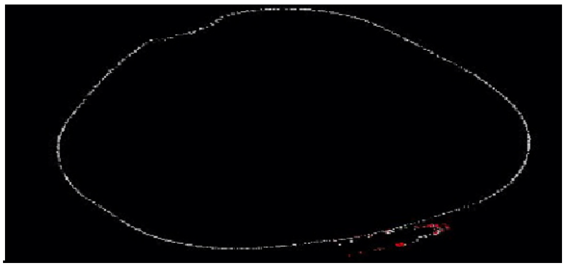


Fig. 4. Slice based feature detection (the object is colored in red)

We start by slicing the digital model horizontally. We average the points between the slices. The curvature scaler descriptor here finds the ratio between the total number of boundary pixels (length) and the number of boundary pixels where the boundary direction changes significantly. The smaller the number of direction changes, the straighter the boundary. In this case, we map the points on the slice to a polar coordinate system because the shape of a body cross-section is in an oval shape.

With *n* points, the coordination transforms are defined by equations (1)-(2):

$$r = \sqrt{(x - \frac{1}{n}\sum_{i=1}^{n} x_i)^2 + (y - \frac{1}{n}\sum_{i=1}^{n} y_i)^2} \tag{1}$$

$$\begin{aligned} \theta &= \arccos\left(\frac{x - \frac{1}{n}\sum_{i=1}^{n} x_i}{r}\right) \quad \text{when} \quad y > \frac{1}{n}\sum_{i=1}^{n} y_i \\ \theta &= \pi + \arccos\left(-\frac{x - \frac{1}{n}\sum_{i=1}^{n} x_i}{r}\right) \quad \text{when} \quad y < \frac{1}{n}\sum_{i=1}^{n} y_i \end{aligned} \tag{2}$$

Then we use a function of the radius and angles of the points to calculate the anomalous features. From Figure 4, we can see that the anomalous increased the point intensity of surface contour. So the intensity-based method calculates the average point distance of specified number of neighboring points. If the neighborhood average distance exceeds the product of overall average distance and specified ratio, then the neighboring local area was marked as the anomalous. The algorithm is presented below:

```
Objective: Determine two sets B and W - set of body points and set of
alien points accordingly, where $B \cup W = P$.
1. Determine the center point and change Cartesian to polar coordinates.
2. Select a start point and sort points by the angle to form arrangement P.
3. from start point in the arrangement by angle $P_i \in P$:
    Calculate the values of each point i:
    $\Delta r_i$ - The difference between two consecutive points.
    $\Delta \alpha_i$ - The angle between tangent-vectors of two consecutive points.
4. Calculate the global average value of $|\Delta r|$.
5. Check each point P for the basis condition to discriminate into body or
alien objects:
    If $k1*Avg(|\Delta r|)<|\Delta r_i|<k3*Avg(|\Delta r|)$ and $\pi/k2<\Delta\alpha_i<\pi/k4$
    Then $P \in B$ else $P \in W$, where k1,k2,k3,k4 is the threshold pre-evaluated.
```

The surface based detection was base on the surface curvature calculation. It is well known how the notion of Gaussian curvature extends to such discrete surfaces S which formed by triangle facets. Thus the Gaussian curvature is supported on the vertices $p \in S$. Its numerical value is the product of the principal curvatures, κ_1 and κ_2, of the given point. From the concept of Discrete Differential Geometry, the curvature is calculated by the following equation:

$$K(p) = \frac{1}{A(p)}\left(2\pi - \sum_{p_i \in N(p)} \theta_i\right) \tag{3}$$

where $A(p)$ is the sum of surface area of triangle facet of the vertex point p, and θ is the angle of the corresponding facet.

The curvature of each point is calculated by the following equation [17]:

$$K(p) = \frac{\dot{x}\cdot\ddot{y} - \ddot{x}\cdot\dot{y}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \tag{4}$$

where $\dot{x}\cdot\dot{y}, \ddot{x}\cdot\ddot{y}$ represented the first-order and second-order differential. For the sequenced discrete points, we use the difference to replace the differential. The interval expressed by curvatures which exceeds the specified threshold illuminates the anomalous. Fig. 5 shows the final detection results.

We believe that fusion of the intensity-based detection and surface based detection will improve the feature detection accuracy and reduce the noise. Because we have the heterogeneous human models with different resolution and different orientations and sizes, model registration appears to be a challenge. However, for the actual human scanning systems, this is not a problem because the coordinates of the point clouds are known to the designers.

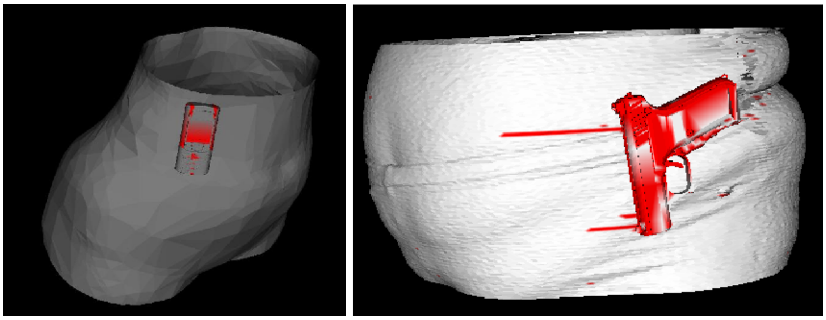


Fig. 5. Detected mobile phone object (left) and highlighted gun with the surface feature (right)

4 Conclusions

The goal of the proposed research is to build a virtual human imaging system for designing and evaluating the related technologies before a physical system is built. Given the available databases of anthropological models from CAESAR, 3D scanners and the physical parameters of human imaging systems, we simulate the scanning imagery data with High Frequency Simulation System (HFSS).

The concealed object detection algorithms are developed based on the wave intensive and surface characteristics. This forward-thinking approach intends to transform the development of human imaging technologies from being device-specific and proprietary to being device-independent and open source-oriented. It will also transform the research into a systematic design process, enabling multi-disciplinary innovations in digital human modeling, computer vision, information visualization and computational aesthetics.

The result of this project would have impacts on privacy-aware imaging systems in airports and medical systems. They can also benefit custom-fit products that are designed from personal 3D scanning data. Our results can be used in the reconstruction of ancient artifacts in digital archeology. In addition, they can be applied to medical diagnoses and procedures, such as virtual colonoscopy.

Acknowledgement

We would like to thank for National Science Foundation for the Cyber Trust grant CT-ER 0716657 and ARO for the research grant for security technologies. Many thanks to Joseph Laws, Emily Hart, Nathaniel Bauernfeind, and Brian Zeleznik for their participation of the project. We would also appreciate the help from Alva Karl of Air Force for the CAESER database.

References

1. Law, J., Cai, Y.: Feature Hiding in 3D Human Body Scans. Journal of Information Visualization 5(4) (2006)
2. Laws, J., Cai, Y.: A Privacy Algorithm for 3D Human Body Scans. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 870–877. Springer, Heidelberg (2006)

3. Cai, Y., et al.: Spatiotemporal data mining for tracking ocean objects. In: Proceedings of IEEE Space Mission Challenges to IT, Pasadena, CA (2006)
4. Cai, Y., et al.: Visual Transform for spatiotemporal data mining. Journal of Knowledge and Information Systems (to appear) (2007)
5. BodySearch imaging system, American Science and Engineering, Inc., 829 Middlesex Turnpike, Billerica, MA 01821 (2007)
6. Secure 1000 imaging system, IRT Corporation, 6020 Cornerstone Court West, San Diego, CA 92121 (2007)
7. McMakin, D.L., Sheen, D.M., Collins, H.D., Hall, T.E., Severtsen, R.H.: Wideband, millimeter-wave, holographic surveillance systems. In: EUROPTO International Symposium on Law Enforcement Technologies: Identification Technologies and Traffic Safety, Munich, FRG, SPIE, vol. 2092, pp. 131–141 (1995)
8. Sheen, D.M., McMakin, D.L., Collins, H.D.: Circular scanned millimeter-wave imaging system for weapon detection. In: EUROPTO International Symposium on Law Enforcement Technologies: Identification Technologies and Traffic Safety, Munich, FRG, SPIE, vol. 2092, pp. 122–130 (1995)
9. McMakin, D.L., Sheen, D.M., Collins, H.D., Hall, T.E., Smith, R.R.: Millimeter-wave, high-resolution, holographic surveillance system. In: EUROPTO International Symposium on Substance Identification Technologies, Innsbruck, Austria, SPIE, vol. 2092, pp. 525–535 (1993)
10. Sheen, D.M., McMakin, D.L., Collins, H.D., Hall, T.E.: Weapon detection using a wideband millimeter-wave linear array imaging technique. In: EUROPTO International Symposium on Substance Identification Technologies, Innsbruck, Austria, SPIE, vol. 2092, pp. 536–547 (1993)
11. Huguenin, G.R., Goldsmith, P.F., Deo, N.C., Walker, D.K.: Contraband Detection System, U. S. Patent 5,073,782 (1991)
12. Browne, J.: MM waves aid commercial applications, Microwaves and RF, pp. 113-116 (July 1992)
13. Goodman, J.W.: Introduction to Fourier Optics. McGraw-Hill, New York (2005)
14. Soumekh, M.: Bistatic synthetic aperture radar inversion with application in dynamic object imaging. IEEE Transactions on Signal Processing 39(9), 2044–2055 (1991)
15. Soumekh, M.: Fourier Array Imaging. Prentice Hall, Englewood Cliffs (1994)
16. Anthropometry Resource (CAESAR), Final Report, Volume I: Summary, AFRL-HE-WP-TR-2002-0169, United States Air Force Research Laboratory, Human Effectiveness Directorate, Crew System Interface Division, 2255 H Street, Wright-Patterson AFB OH 45433-7022 and SAE International, 400 Commonwealth Dr., Warrendale, PA 15096 (2007)
17. Jalba, A.C., Wilkinson, M.H.F., Roerdink, J.: Shape Representation and Recognition Through Morphological Curvature Scale Spaces. IEEE Trans. Image Processing 15(2), 331–341 (2006)
18. Forsyth, D.A., Fleck, M.M.: Automatic detection of human nudes. International Journal of Computer Vision 32(1), 63–77 (1999)
19. Forsyth, D.A., Fleck, M.M.: Body Plans. In: Proc. CVPR 1997, pp. 678–683 (1997)
20. Forsyth, D.A., Fleck, M.M.: Identifying nude pictures. In: Proceeding Third IEEE Workshop on Applications of Computer Vision, pp. 103–108 (1996)
21. Goldgof, D.B., Huang, T.S., Lee, H.: Feature extraction and terrain matching. In: Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognition, Ann Arbor, MI (May 1988)
22. Goldgof, D.B., Huang, T.S., Lee, H.: A Curvature-Based Approach to Terrain Recognition, vol. 11(11), pp. 1213–1217 (November 1989)

23. Gordon, G.: Face recognition based on depth and curvature features. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Champaign Illinois), pp. 108–110 (1992)
24. Li, P., Corner, B.D., Paquette, S.: Evaluation of a surface curvature based landmark extraction method for three dimensional head scans. In: International Ergonomics Conference, Seoul (2003)
25. Liu, X., Kim, W., Drerup, B.: 3D Characterization and Localization of Anatomical Landmarks of the Foot. In: Proceeding (417), Biomedical Engineering. Acta Press (2004)
26. Fleck, M.M., Forsyth, D.A., Bregler, C.: Finding naked people. In: Buxton, B., Cipolla, R. (eds.) ECCV 1996. LNCS, vol. 1065, pp. 593–602. Springer, Heidelberg (1996)
27. Ratner, P.: 3-D human modeling and animation. John Wiley & Sons, Inc., Chichester (2003)
28. Robinette, K.M., Blackwell, S., Daanen, H.A.M., Fleming, S., Boehmer, M., Brill, T., Hoeferlin, D., Burnsides, D.: Civilian American and European Surface Anthropometry Resource (2002)
29. Ioffe, S., Forsyth, D.A.: Probabilistic methods for finding people. International Journal of Computer Vision 43(1), 45–68 (2001)
30. Sonka, M., et al.: Image processing, analysis and machine vision, PWS Publishing (1999)
31. Suikerbuik, C.A.M.: Automatic Feature Detection in 3D Human Body Scans. Master thesis INF/SCR-02-23, Institute of Information and Computer Sciences. Utrecht University (2002)
32. Suikerbuik, R., Tangelder, H., Daanen, H., Oudenhuijzen, A.: Automatic feature detection in 3D human body scans. In: Proceedings of SAE Digital Human Modeling Conference, 04-DHM-52 (2004)
33. Mathworks MRI Phantom (2008), `http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=1759&objectType=file`
34. NLM, Visible Human Project (2008), `http://www.nlm.nih.gov/research/visible/visible_human.html`
35. DARPA Virtual Soldier (2008), `http://www.wired.com/news/medtech/0,1286,60016,00.html`
36. Neill, D.B., Moore, A.W.: Anomalous spatial cluster detection. In: Proc. KDD 2005 Workshop on Data Mining Methods for Anomaly Detection, pp. 41–44 (2005)
37. Neill, D.B., Moore, A.W.: Rapid detection of significant spatial clusters. In: Proc. 10th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining, pp. 256–265 (2004)
38. Salvador, S., Chan, P.: Fastdtw: Toward accurate dynamic time warping in linear time and space. In: KDD Workshop on Mining Temporal and Sequential Data (2004)
39. Shyu, M.L., Chen, S.C., Sarinnapakorn, K., Chang, L.W.: A novel anomaly detection scheme based on principal component classifier. In: Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop (2003)
40. Zhang, J., Zulkernine, M.: Anomaly Based Network Intrusion Detection with Unsupervised Outlier Detection. In: Symposium on Network Security and Information Assurance – Proc. of the IEEE International Conference on Communications (ICC), Istanbul, Turkey (June 2006)
41. Burbeck, K., Tehrani, S.N.: ADWICE: Anomaly Detection with Real-time Incremental Clustering. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 407–424. Springer, Heidelberg (2005)

42. Gomez, J., Gonzalez, F., Dasgupta, D.: An Immuno-Fuzzy Approach to Anomaly Detection. In: the proceedings of the 12th IEEE International Conference on Fuzzy Systems (FUZZIEEE), May 25-28, 2003, vol. 2, pp. 1219–1224 (2003)
43. Wise, J.A., Thomas, J.J., Pennock, K., Lantrip, D., Pottier, M., Schur, A., Crow, V.: Visualizing the non-visual: spatial analysis and interaction with information from text documents. In: Proceedings of the 1995 IEEE Symposium on Information Visualization, Atlanta, Georgia, October 30-31, 1995, p. 51 (1995)
44. Keller, P., McMkin, L., Sheen, D., McKinnon, A., Summet, A.J.: Privacy Algorithm for Cylindrical Holographic Weapons Surveillance Systems. In: Proc. SPIE, Applications and Science of Computational Intelligence III, vol. 4055, pp. 476–483 (2000)
45. Sheen, D.M., et al.: Concealed explosive detection on personnel using a wideband holographic millimeter-wave imaging system. In: Proceedings of the SPIE, AEROSENSE Conference, Orlando, FL, vol. 2755 (1996)

Interactive Visualization of Network Anomalous Events

Yang Cai and Rafael de M. Franco

Carnegie Mellon University
ycai@cmu.edu
www.cmu.edu/vis

Abstract. We present an interactive visualization and clustering algorithm that reveals real-time network anomalous events. In the model, glyphs are defined with multiple network attributes and clustered with a recursive optimization algorithm for dimensional reduction. The user's visual latency time is incorporated into the recursive process so that it updates the display and the optimization model according to a human-based delay factor and maximizes the capacity of real-time computation. The interactive search interface is developed to enable the display of similar data points according to the degree of their similarity of attributes. Finally, typical network anomalous events are analyzed and visualized such as password guessing, etc. This technology is expected to have an impact on visual real-time data mining for network security, sensor networks and many other multivariable real-time monitoring systems.

Keywords: interaction, visualization, network anomaly, anomalous event, clustering.

1 Introduction

The paradigm of data visualizations has shifted from merely visual data rendering to the model-based visual analysis [1-9][29-31]. Examples include: 1) *graph models*, such as the social interaction theory of "the six degrees of separation" [26], the "power law of the linked interactions [22], *minimal graph cuts* for analyzing the outliers in a very large social network [23], 2) *color models* such as the spectrograph [24] of the interaction patterns emerged from gas stations and cellular phone towers, 3) the *geographical profiling model* for investigating serial killer's spatio-temporal patterns [27], 4) the cellular automata model for simulating the dynamics of mass panic in public places[25]. These methods computationally incorporate modeling, rules and visualization in one algorithm, which enables emergent pattern discovery and empirical experimentation.

However, there are limitations to these existing approaches: 1) they are 'off-line' models, which are based on isolated databases without connections to other dynamic systems or continuously updated data streams; 2) they do not normally take physical interactions into account, e.g. location, duration and field strength; and 3) many visual analytics for network conditions are static, one-shot, rather than interactive or iterative.

In this study, we focus on the following scientific problems: First, we develop a real-time network event simulator with realistic data streams for visual analytics. We construct a synthetic real-time network database from a real network server and historical databases. To build such a dynamic database itself warrants scientific research because it involves privacy, fidelity and bandwidth issues. We explore several ways to downsize the data space in orders of magnitude. We then investigate which method is the best for this kind of problem. We start with a shape-based dynamic clustering model for massive multi-attribute data points (e.g. up to 64 attributes in 10,000 data points) in a 2D space. We then experiment with other representations, such as colored micro-arrays, pixel maps and 3D surfaces, etc.

Secondly, we develop interactive visualization algorithms. Most of existing visualization models that we found are non-interactive, and follow the sequence of 'data-process-display'. In this study we develop a set of algorithms that incorporate human latency, manual navigation and affection. For example, the human latency-aware algorithm converts a batch optimization problem to an incremental optimization problem that enables the computer to visualize more network data by an order of magnitude. The manual navigation of data would allow the multi-resolution system to 'hide' unnecessary data in a peripheral area and only display the user interested data at the highest resolution. As a result, the visualization system can handle increased network data by orders of magnitude.

Finally, we study the visual detection of signatures and anomalies. The biggest issue of network security is intrusion detection and recognizing whether a system has been compromised. There are two groups of methods: signature detection and anomaly detection [13-19]. Signature-based threat detection scans network traffic for a set of predefined attack or vulnerability patterns – similar to today's anti-virus checking. It seeks the "known bad" signatures and assumes that everything else is good. On the other side, the behavior-based anomaly detection methods try to define the "known good" or "normal" behaviors and assume any deviations from the normal behavior are possible attacks. There are many pros and cons in these two approaches: signature detection won't detect any undefined problems and it is computationally expensive when the signature definitions increase; the anomaly detection methods can potentially detect unknown attacks, however, they often lead to false alarms because it is not so easy to define normal patterns. In our study, visualization is actually a detection and learning tool that combines signature detection and anomaly detection seamlessly. Compared to those typical Network Intrusion Detection Systems (NIDS), which normally require a lengthy supervised machine learning process and a large volume of historical data, our approach can avoid this preparation and directly perform the learn-by-doing detection. Here we integrate the signature detection for known attack patterns (e.g. password guessing) and anomaly detection for spatial patterns (e.g. shape anomaly) or temporal patterns (e.g. periodicity). This provides a visual way to interpret the network flow and gives a human expert the possibility to make the final decision on the detection of an anomaly.

2 The Real-Time Network Event Simulator

We assume at any given time, the visualization system assimilates a real-time sequence of a network data set, up to 10,000 points, in which each of them has up to 64 attributes. The update interval is within 10 seconds. The key factor in this project is

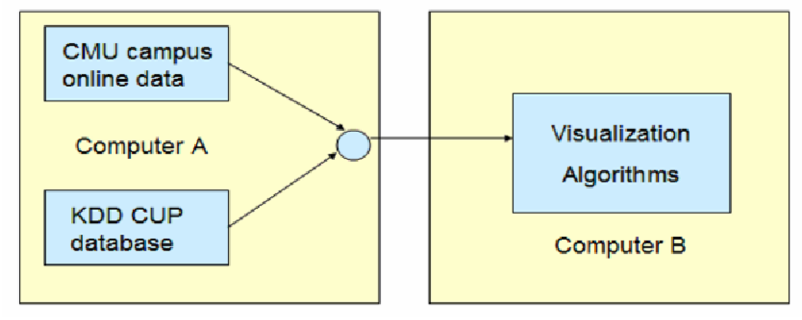


Fig. 1. The architecture of the network data simulator and visualization system

how to obtain a stream of continuous network data in a realistic and non-invasive way. We use two data sources: 1) CMU campus real-time database, and 2) the KDD CUP 1999 database [11].

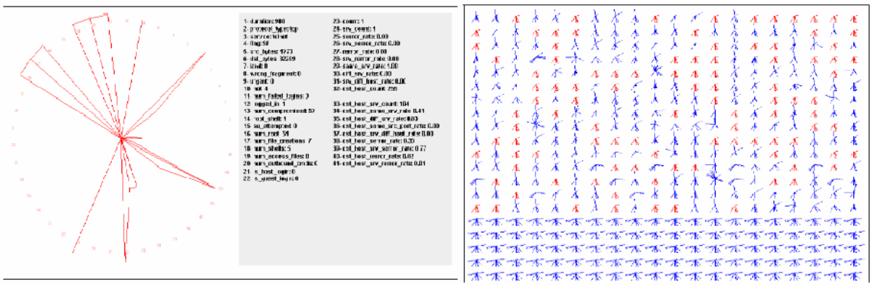
To simulate the dynamics of the network data flow, we use two computers (A as the data source and B as the visualization terminal) to generate the simulated continuous network data. Since 2005, we have been collecting the real-time network data from the Andrew network server in CMU at 1 point per 10 second interval. The database is in XML format. We have developed a Java code to automatically capture the data so that personal identifications are removed. The CMU campus live database gives us the real-world fresh data. Every 10 seconds, computer A passes the data to computer B. The visualization software updates accordingly. To effectively use the information, we add necessary software tools on Computer A, such as the '*tcpdump*' and network traffic analysis utilities.

Since network attacks are not frequent, we also use the captured attack data from the KDD CUP 1999 database [11] as a key reference for designing the algorithm. This data represents thousands of connections with 41 different attributes. There are 2 kinds of attributes: continuous and discrete.

3 Glyph-Based Dynamic Clustering and Visualization

The pre-processing task is to normalize the continuous data so that each attribute can have the same level of influence when comparing one dataset to another (calculating the Euclidean distance). The normalization is performed dividing each attribute by the maximum attribute's value of the whole data scanned during a period of time. The normalized attributes can also be multiplied by a coefficient weight that can be adjusted according to the importance of each attribute for the detection of an anomaly.

The second step of the algorithm is to visualize each network connection and their 41 attributes on a 2 Dimensional graphic. To resolve that, it is necessary to find a visualization technique that represents a multidimensional (n-D) array on a 2-D graphic. The star glyphs technique [12] is an elegant solution for the problem and it fits perfectly for this application. In the glyph's plot, the dimensions are represented as equal-spaced angles from the center of a circle and each axis represents the value of the dimension. Figure 2 illustrates a glyph using network attributes.



1- duration: 988
2- protocol_type: tcp
3- service: telnet
4- flag: SF
5- src_bytes: 1773
6- dst_bytes: 32509
7- land: 0
8- wrong_fragment: 0
9- urgent: 0
10- hot: 4
11- num_failed_logins: 3
12- logged_in: 1
13- num_compromised: 57
14- root_shell: 1
15- su_attempted: 0
16- num_root: 54
17- num_file_creations: 7
18- num_shells: 5
19- num_access_files: 0
20- num_outbound_cmds: 0
21- is_host_login: 0
22- is_guest_login: 0
23- count: 1
24- srv_count: 1
25- serror_rate: 0.00
26- srv_serror_rate: 0.00
27- rerror_rate: 0.00
28- srv_rerror_rate: 0.00
29- same_srv_rate: 1.00
30- diff_srv_rate: 0.00
31- srv_diff_host_rate: 0.00
32- dst_host_count: 255
33- dst_host_srv_count: 104
34- dst_host_same_srv_rate: 0.41
35- dst_host_diff_srv_rate: 0.03
36- dst_host_same_src_port_rate: 0.00
37- dst_host_srv_diff_host_rate: 0.00
38- dst_host_serror_rate: 0.35
39- dst_host_srv_serror_rate: 0.77
40- dst_host_rerror_rate: 0.02
41- dst_host_srv_rerror_rate: 0.01

Fig. 2. Glyph definition (left) and the fixed position glyph display (right)

The right image in Fig. 2 shows 400 network connections displayed in a Glyph form. The glyphs highlighted in red are connections that have similar attributes and consequently a similar glyph's form. The glyphs on the bottom are connections from a DoS (Denial of Service) attack and, comparing to all the other connections, the abnormal form emerges.

The clustering process has two main purposes: first to reduce the large amount of data connections (that can be more than 10 thousand for big networks) and second to detect a possible anomaly. In order to understand how clustering data can perform these two tasks, it is intuitively better to imagine the clusters as galaxies and the network connections as planets. Each planet has its own characteristics that are given by its attributes. Thus, planets with similar characteristics cluster together and form a galaxy. On normal network traffic, it is expected to have a large number of data glyphs with the same normal pattern. This data will create a cluster (like planets of a same galaxy). However, for the case of a network intrusion, different clusters will be created (like a far-away galaxy). These new clusters do not have the same patterns as normal data, thus they are distant from the normal cluster. If the distance between this cluster and the "normal" data is bigger than a certain threshold, the algorithm recognizes it as an anomaly. The creation of a cluster was implemented with a simple algorithm by comparing the similarity among the connections' attributes. This comparison can be implemented by calculating the distance between two connections. The Euclidean distance is the simplest form to measure the distance between two vectors and it is applied.

The first step is to create a cluster using a first data called "reference". The Euclidean distance between this "reference" and the rest of the data is calculated. Data that have a distance smaller than a threshold are added to the cluster and set with a flag telling that they were already selected. The loop continues choosing new unselected "reference" data and so on. The cluster's mean is also calculated during the process and used in the detection part. The anomaly detection for this algorithm is quite simple, and it should be improved in future work. The detection is made by adding an anomaly threshold: if the Euclidean distance between a cluster's mean and the global mean from all the clusters is bigger than the anomaly threshold, all the data from this cluster are anomalies.

We start with a case study of network anomaly visualization. We display clusters of 10,000 data points, in which each point has 64 attributes in real-time on a regular PC. We test Principal Component Analysis, Kohonen Self Organizing Maps (SOMS) and Multidimensional Scaling (MDS). The MDS algorithm is to organize a multidimensional data on a two dimensional graphic by coordinate pairs(x,y). The cartesian plane makes axes explicit and it is easier to organize data according to their characteristics. The idea of the MDS is to measure all the data distance in an N-dimensional space and place it on 2D display so they obey a same mutual distance relationship. However, a 2D perfect configuration is not always possible. Let d_{ij} be the multidimensional distance between a point i and j, calculated by the Euclidean distance. Let also d_{ij} be the 2-dimensional distance between the same point i and j calculated with the Pythogorean Theorem $\delta_{ij} = \sqrt{x^2 + y^2}$. If $d_{ij} \neq \delta_{ij}$ than there is a stress between the data and 2D representation. The stress looks to the multidimensional and 2-dimensional distances between the point P_1 and P_2. The stress is calculated and the 2D positions have to be placed on a way that it minimizes this stress. To minimize it, we applied the simplex optimization algorithm.

$$stress = \sum_{i<j} \left(\frac{d_{ij} - \delta_{ij}}{d_{ij}} \right)$$

Figure 3 is an example of the latency-aware algorithm. The glyphs in red are the data from a highlighted cluster. The glyphs in blue and green are the other clusters organized with the MDS algorithm.

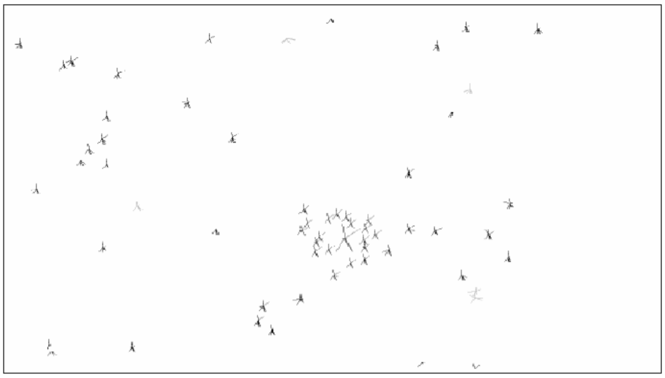


Fig. 3. Example of the clustering algorithm

4 Interactive Visualization Algorithms

Here we develop a set of algorithms that incorporate human latency, manual navigation and affection. For example, the human latency-aware algorithm converts a batch optimization problem into an incremental optimization problem that enables the computer to visualize more network data by an order of magnitude. The manual navigation of data would allow the multi-resolution system to ‘hide’ unnecessary data in a peripheral area and only display the user interested data at the highest resolution. As a result, the visualization system can handle more network data by multiple orders of magnitude. Furthermore, in a 24/7 pervasive real-time visualization system, psychological boredom is inevitable phenomena.

4.1 Latency-Aware Computing

Human vision has about 0.1 second latency which has been an important factor in modern video and movie technologies [10]. In principle, a system need not update data faster than a human's response time. In light of this, we can use the human latency to design many novel human-centric computing algorithms that incorporate the latency factor. Many visualization methods involve the time-consuming algorithms for clustering and optimization. Instead of waiting for minutes to see the updated results, the latency-aware algorithms are designed to synchronize the speed of human vision by incremental updating. This should create a new philosophy for algorithm design.

In this study, we develop the latency-aware multidimensional scaling model. The algorithm is to organize a multidimensional data on a two dimensional graphic by coordinate *pairs(x,y)*. Described in Section 3, according to the Pythogorean Theorem. The stress looks to the multidimensional and 2-dimensional distances between the points. The stress is calculated and the 2D positions have to be placed on a way that it minimize this stress. To minimize it, we applied the simplex optimization algorithm. The only difference is that here we use an incremental update with a visual pulse interval (e.g. say, between 1/25 sec to 10 sec). Faster than 1/25 second is not necessary because human eyes can't catch up. Slower than a 10 sec interval might cause anxiety.

4.2 Search for Similar Patterns

Human eyes are good for recognition but poor in searching a massive numerical matrix. On the other hand, a computer is poor in recognition but efficient in performing a large searching task. Here we develop an interactive signature or anomaly search algorithm: Given a sample glyph with a certain patterns, the computer will search all the data on the screen and highlight the ones with similar patterns. Figure 4 shows an illustration of the results.

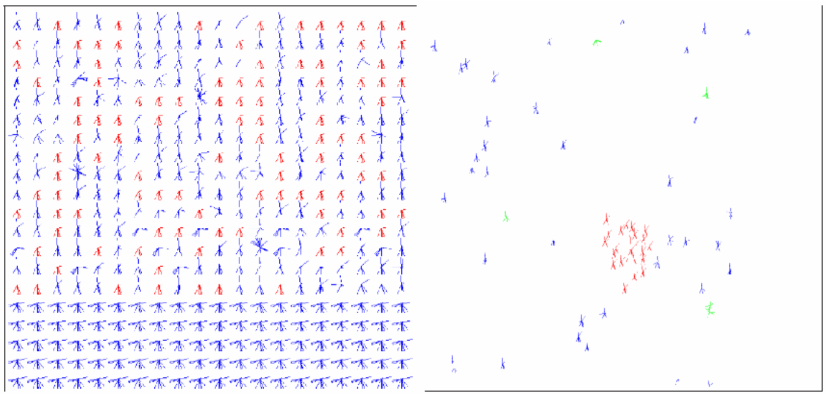
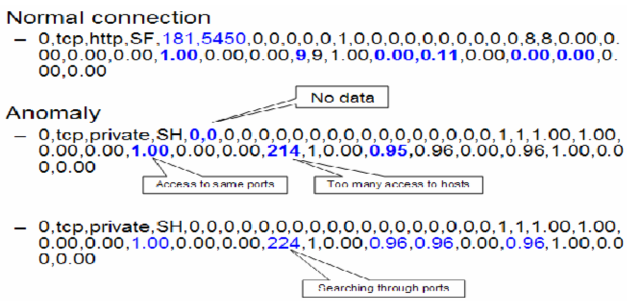


Fig. 4. Highlighted (in red) anomalies with the fixed glyphs (at the bottom of the left image) and dynamically clustered ones (red glyphs in the right image)

5 Visual Transformation for Signature Detection

Signature detection is actually a rule-based expert system, which scans network traffic for a set of predefined attack patterns – similar to today's anti-virus checking. It seeks the "known bad" signatures. With the help of technologies like NIDS (Network Intrusion Detection Systems), Sniffers and SNMP (simple network management protocol), different kinds of network attributes and thousands of data samples can be retrieved in a very short period of time. However, raw numeric data is difficult to interpret. The first scenario focuses on general normal data generation. The connection and data are made from host B to host A services (HTTP, Telnet, FTP, SMTP, HTTPS). We built several connections. Several connections are recorded with '*tcpdump*'. Figure 5 shows a sequence of a normal dataset and two anomalous datasets.


No data
Access to same ports
Too many access to hosts
Searching through ports

Normal connection

```
– 0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.
  00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.
  00,0.00
```

Anomaly

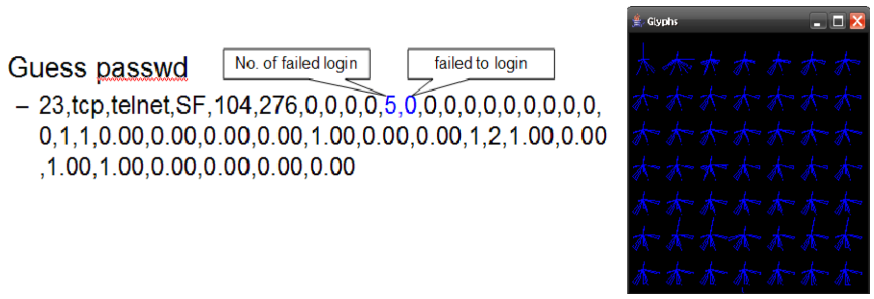
```
– 0,tcp,private,SH,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1.00,1.00,
  0.00,0.00,1.00,0.00,0.00,214,1,0.00,0.95,0.96,0.00,0.96,1.00,0.0
  0,0.00

– 0,tcp,private,SH,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1.00,1.00,
  0.00,0.00,1.00,0.00,0.00,224,1,0.00,0.96,0.96,0.00,0.96,1.00,0.0
  0,0.00
```

Fig. 5. Samples of normal string vs. anomalous strings

The second scenario is to make forged data and send out lots of attack data and abnormal data from host A, and then record them with '*tcpdump*'. The typical case for creating abnormal data is to employ *'ipmagic*', with which most IP packets can be manipulated in any way that we want. Following is a command to send a single identical packet for Land attack, which consists of same departing address and port to the destination address and port. It covers all attack groups, where DARPA set the rules to define attack categories. They are categorized with 4 groups below. 1) DoS: denial-of-service, 2) R2L: remote machine to local Machine, 3) U2R: user to root, and 4) Probing: port scanning. For each group, a representative attack is conducted: *'juno'* attack for DoS, 'login' password guessing for R2L, and the attempt for U2R and *'nmap'* scanning for probing attack. All the attack runs on the network are stored.

The next step is to replay stored data in a continuous way where Linux scripts are used continuously. After source data is all set, by running *'tcpreplay*' with options and Linux's scripting, data can be sent in very diverse forms with source data. Following is an example case to send the *'dumped_data'* with a form of randomized IP address and looped 100 times through the interface. After filtering, we can distract data from *'tcpdumped'* data into the computer network. This data is put into visualization software, and then the software makes moving glyphs. Figure 6 shows an example of the anomalous event '*password guessing.*'



Guess passwd

– 23,tcp,telnet,SF,104,276,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1,2,1.00,0.00,1.00,1.00,0.00,0.00,0.00,0.00

Fig. 6. Example of 'password guessing' attack

This algorithm uses the Euclidean distance among the network connections in order to create different clusters according to their patterns. Based on the distance among these clusters, anomalies can be detected. It is also possible to visualize the multidimensional data using the star glyphs technique. These glyphs give a visual representation of the numerical attributes, which allows a human to visually detect abnormalities by just looking at the different forms of glyphs.

6 Conclusions

Here we present an interactive visualization and clustering algorithm that reveals real-time network anomalous events. We first build a real-time network event simulator based on the replay of the collected network log data from multiple sources. Then we develop the interactive visualization model that can be connected to the simulator. In the model, glyphs are defined with multiple network attributes and clustered with the recursive optimization algorithm for dimensional reduction. The user visual latency time is incorporated into the recursive process so that it updates the display and the optimization model according to the human vision delay factor and maximizes the capacity of the real-time computation. The interactive search interface is developed to enable the display of similar data points according to their similarity in attributes. Finally, typical network anomalous events are analyzed and visualized such as password guessing, etc.

This technology is expected to have an impact on visual real-time data mining for network security, sensor networks and many other multivariable real-time monitoring systems.

Acknowledgement

The authors would like to thanks the sponsors for this project: US Army Research Office (ARO) and Korea Information Security Agency (KISA). In addition, the authors are grateful to the help from Brian Zeleznik for his comments and editing.

References

1. Cai, Y. (ed.): Ambient Intelligence for Scientific Discovery. LNCS (LNAI), vol. 3345. Springer, Heidelberg (2005)
2. Cai, Y., Abscal, J. (eds.): Ambient Intelligence for Everyday Life. LNCS (LNAI), vol. 3864. Springer, Heidelberg (2006)
3. Cai, Y., Terrill, J.D.: Visual Analysis of Human Dynamics. Journal of Information Visualization 5(4) (2006)
4. Cai, Y.: Ambient Intelligence for Knowledge Discovery: Editorial. To appear on International Journal of Human-Computer Studies (2007)
5. Cai, Y., et al.: Visual Transform for Spatiotemporal Data Mining. Accepted and to appear on Journal of Knowledge and Information Systems (KAIS) (2009)
6. Laws, J., Cai, Y.: Feature Hiding for Human 3D Scan Data. Journal of Information Visualization 5(4) (2006)
7. Cai, Y., Snel, I., Bharathi, B.S., Klein, C., Seetharaman, J.K.: BioSim: A Character-Based Biomedical Problem Solving Environment. Journal of Future Generation Computer Systems 21(7), 1145–1156 (2005)
8. Cai, Y.: Minimalism Context-Aware Displays. Journal of CyberPsychology and Behavior 7(3) (2004)
9. Tanz, O., Shaffer, J.: Wireless local area network positioning. In: Cai, Y. (ed.) Ambient Intelligence for Scientific Discovery. LNCS (LNAI), vol. 3345, pp. 248–262. Springer, Heidelberg (2005)
10. Boff, K.R., Kaufman, L., Thomas, J.P. (eds.): Human Performance Measures Handbook. Wiley and Sons, Chichester (1986)
11. KDD CUP 1999 data (1999), `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
12. Matthew Ward. Mavis home page (1996), `http://web.cs.wpi.edu/~matt/research/MAVIS/`
13. Neill, D.B., Moore, A.W.: Anomalous spatial cluster detection. In: Proc. KDD 2005 Workshop on Data Mining Methods for Anomaly Detection, pp. 41–44 (2005)
14. Neill, D.B., Moore, A.W.: Rapid detection of significant spatial clusters. In: Proc. 10th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining, pp. 256–265 (2004)
15. Salvador, S., Chan, P.: Fastdtw: Toward accurate dynamic time warping in linear time and space. In: KDD Workshop on Mining Temporal and Sequential Data (2004)
16. Shyu, M.L., Chen, S.C., Sarinnapakorn, K., Chang, L.W.: A novel anomaly detection scheme based on principal component classifier. In: Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop (2003)
17. Zhang, J., Zulkernine, M.: Anomaly Based Network Intrusion Detection with Unsupervised Outlier Detection. In: Symposium on Network Security and Information Assurance – Proc. of the IEEE International Conference on Communications (ICC), Istanbul, Turkey (June 2006)
18. Burbeck, K., Tehrani, S.N.: ADWICE: Anomaly Detection with Real-time Incremental Clustering. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 407–424. Springer, Heidelberg (2005)
19. Gomez, J., Gonzalez, F., Dasgupta, D.: An Immuno-Fuzzy Approach to Anomaly Detection. In: The proceedings of the 12th IEEE International Conference on Fuzzy Systems (FUZZIEEE), May 25-28, 2003, vol. 2, pp. 1219–1224 (2003)
20. Levin, I.: KDD 1999 Classifier Learning Contest: LLSoft's Results Overview. In: ACM SIGKDD Explorations 2000, pp. 67–75 (January 2000)

21. Pfahringer, B.: Winning the KDD 1999 Classification Cup: Bagged Boosting. In: ACM SIGKDD Explorations 2000, pp. 65–66 (January 2000)
22. Barabasi, A.L.: Linked: The New Science of Networks, Perseus (2002)
23. Cowell, A., et al.: Understanding the Dynamics of Collaborative Multi-Party Discourse, IVJ, 5(4) (2006)
24. Chakrabarti, D., Zhan, Y., Blandford, D., Faloutsos, C., Blelloch, G.: NetMine: New Mining Tools for Large Graphs. In: The SDM 2004 Workshop on Link Analysis, Counter-terrorism and Privacy (2004)
25. Eagle, N., Pentland, A.: Reality Mining: Sensing Complex Social Systems. Personal and Ubiquitous Computing (September 2005)
26. Helbing, D., Farkas, I.J., Vicsek, T.: Simulating dynamical features of escape panic. Nature 407, 487–490 (2000)
27. Guare, J.: Six Degrees of Separation, Vintage (1990)
28. Rossmo, K.: Geograpical Profiling. CRC Press, Boca Raton (1990)
29. NML, Visible Human Project (2008), `http://www.nlm.nih.gov/research/visible/visible_human.html`
30. Wong, P.C., Rose, S.J., Chin Jr., G., Frincke, D.A., May, R., Posse, C., Sanfilippo, A., Thomas, J.: Walking the Path - A New Journey to Explore and Discover through Visual Analytics, IVJ, 5(4) (2006)
31. Beale, R., Hendley, B., Pryke, A., Wilkins, B.: Nature-inspired Visualisation of Similarity and Relationships in Human Systems and Behaviours. IVJ 5(4) (2006)

Numerical Algorithms

Towards Low-Cost, High-Accuracy Classifiers for Linear Solver Selection*

Sanjukta Bhowmick, Brice Toth, and Padma Raghavan

Department of Computer Science and Engineering,The Pennsylvania State University, University Park, PA 16802
bhowmick@cse.psu.edu,
btoth@cse.psu.edu,
raghavan@cse.psu.edu

Abstract. The time to solve linear systems depends to a large extent on the choice of the solution method and the properties of the coefficient matrix. Although there are several linear solution methods, in most cases it is impossible to predict apriori which linear solver would be best suited for a given linear system. Recent investigations on selecting linear solvers for a given system have explored the use of classification techniques based on the linear system parameters for solver selection. In this paper, we present a method to develop low-cost high-accuracy classifiers. We show that the cost for constructing a classifier can be significantly reduced by focusing on the computational complexity of each feature. In particular, we filter out low information linear system parameters and then order the remaining parameters to decrease the total computation cost for classification at a prescribed accuracy. Our results indicate that the speedup factor of the time to compute the feature set using our ordering can be as high as 262. The accuracy and computation time of the feature set generated using our method is comparable to a near-optimal one, thus demonstrating the effectiveness of our technique.

1 Introduction

The solution of sparse linear systems comprises the most time-consuming portion of many scientific simulations. There exist many classes of linear solvers, e.g., direct [7], iterative [2], and multigrid [18] methods. Preconditioning techniques are used to improve the convergence of iterative methods, leading to an even greater number of solver-preconditioner combinations (henceforth called solvers). Earlier research [10, 19] indicates that no linear solution method is consistently the best across different linear systems. Even with a single application, the "best" solution method can vary across time-steps and nonlinear iterations thereby making solver selection a very challenging problem. However, using appropriate solvers can significantly reduce the simulation time of the application [4, 10, 19].

Recent research concerns investigating the application of machine learning techniques [3, 6, 13, 14, 22] to predict efficient solvers. Classifiers are trained using a sample

* This work was supported by grants CCF-0830679 and OCI-0720749 of the National Science Foundation.

dataset of linear systems and associated solvers. Each entry in the sample dataset consists of (i) the set of linear system properties (feature set), (ii) a solver and (iii) the solution criteria. In its simplest version, the machine learning algorithm builds a binary classifier. Given a new linear system (not in the database), a solver and a solution criteria (represented in the database), the classifier predicts whether or not the solver solves the linear system according to the given solution criteria.

Earlier work in this area [13, 14, 22] focuses primarily on generating high-accuracy classifiers and the problem of feature selection is not addressed. Results from [11] demonstrate that filtering out zero-variance features can reduce classifier construction time. These results indicate the need for further investigation on the choice of features and their impact on classifier accuracy and time for construction.

In this paper, we present a method for generating low-cost, high-accuracy classifiers. Section 2 contains a brief overview of classification methods. Sections 3 and 4 contain our main contributions including an ordering scheme to select features and an empirical evaluation of its effectiveness. Section 5 contains brief concluding remarks and future research directions.

2 Background: Supervised Learning and Feature Selection

Supervised learning involves designing a classification function based on a set of already classified data. Often *k-fold cross-validation* is used to improve the accuracy of the classifiers. The database is divided into k subsets. At each instance of learning, k-1 subsets are used as a *training set* , and the remaining subset is used as a *testing set*. The training set is used to build the classifier and the testing set is used to verify the accuracy of the classifier. This process is repeated k times, each time with a different subset as the testing set. The results are combined to produce the final classifier whose accuracy is measured by the percentage of correctly predicted entries in the testing sets [12, 21].

We consider an entry to be represented by a feature set $f_1, f_2, \ldots, f_n$. The objective of a binary classifier is to determine whether an unknown entry E belongs to group $C = 0$ or $C = 1$. Some common supervised learning techniques are given below,
K Nearest Neighbor (NN(k)): In this method the class of entry E is determined according to the class of the majority of its k nearest neighbors in the feature space. The NN(k) algorithm is sensitive to the value of k, usually the higher the value, the better the classification.

Alternating Decision Trees (ADT): In the decision tree algorithm, each node in the tree represents a particular feature. At each point of traversing the tree, a classification decision is made based on the value of that feature in the unclassified entry. A decision tree with one root is known as a Decision Stump (DS). Alternating decision trees represent a weighted and more generalized form of decision trees containing predictors as well as decision nodes.The combined score of the visited predictor nodes and the final leaves determines the classification of the entry. The larger the decision tree, more accurate the classification.

Naive Bayes (NB): This method uses Bayes' theorem to classify the input vector. A naive bayes classifier assumes that the features defining the input vector are independent of each other. The classification can be expressed as; NB(E)= $\frac{p(C=0)}{p(C=1)} \Pi_i^n \frac{p(f_i|C=0)}{p(f_i|C=1)}$; where $p(f_i|C=c)$ is the probability that f_i occurs given that classification is c.

Support Vector Machines (SVM): This method is based on creating a separating hyperplane that maximizes the distance between the two classes in the feature space. The support vector machine classifier can be formulated as; select w, b to minimize $\|w\|$, such that $c_i(w \cdot x_i - b) \geq 1$, for $1 \leq i \leq n$, where w is the vector perpendicular to the separating hyperplane.

Feature selection is a preprocessing step in machine learning for eliminating redundant, noisy and irrelevant data [15], thereby reducing classification time and improving prediction accuracy. Steps for feature selection consist of generating a subset of the features, evaluating its effectiveness in improving classification, and once the final subset is determined, validating it across different data sets [17].

Feature selection methods can be broadly categorized as (i) the *filter model*, where a simple evaluation technique or filter is used to identify the feature subset (ii) the *wrapper model*, where the feature subset model is based on evaluation over a data mining algorithm [16] and (iii) the *hybrid model* which uses both an independent measure (filter) as well as a datamining algorithm for feature selection.

3 Towards Low Cost Feature Selection

The evaluation of linear system features represents a significant cost in generating the classifiers. The cost of computing a feature varies from being proportional to the size of the matrix to being even more expensive than solving the linear system. Properties related to matrix sparsity, structure and the norms typically exhibit computational complexity of $O(n)$ or $O(nnz)$, for a n by n matrix with nnz nonzeros and can be calculated to their exact values. However, spectral properties such as the singular values, eigenvalues or the condition number can be even more expensive to calculate than solving the linear system itself and can be approximated at best.

The cardinality of the feature set contributes to the cost of constructing and using the classifier. Low cardinality sets can be obtained by eliminating low information or redundant features. Low information features are those that do not contribute to the predictive ability of the classifier. Redundant features are those which, based on certain matrix properties, might have the same value and need not be computed multiple times. Since many linear systems are generated and solved as part of a larger simulation process, the properties of the matrix may not be known beforehand. Additionally, scientists using the classifier may lack the requisite expertise to eliminate the redundant features.

Earlier research in feature selection concerns eliminating features of zero variance since they do not contribute to the classifier prediction and identifying redundant features based on loading vectors [11]. This technique of eliminating redundant features, however, can be as expensive as generating an SVM classifier. In this paper, we have extended the definition of low-information features to those whose variance is below a certain threshold, α, not necessarily 0 (as in [11]). We also avoid identifying redundant features based on loading vectors and instead propose inexpensive techniques based on the variance and mean of the feature distribution. Additionally we propose a feature ordering scheme based on the computational complexity of feature calculation, towards constructing a high-accuracy classifier at low cost.

Our two step method is specified below. In the first step, we eliminate low-information features based on statistics of feature distribution, and in the second step we order the remaining features in increasing order of computational complexity to obtain a low-cost feature set of low cardinality.

1. Feature Set Reduction: The predictive power of a feature depends on the distribution of its values across the dataset. Consider two extreme cases of distribution of values. When the values of a feature remain constant across all entries in the dataset, no extra information can be obtained from that property. At the other end, if the number of unique values of a feature is equal to the number of entries, we can base our predictions on just that one property.

Thus the variance across the values of an individual feature provides a good estimate of the statistical dispersion. If a set of feature vectors exhibit *exactly* the same variance and the same mean, it is extremely likely that any one feature from the set can be expressed as a linear function of the others. We can thus select any *one* of these features and eliminate the other redundant feature vectors.

2. Feature Set Ordering: The cardinality of the feature set represents the potential dimensionality of the data space. As we increase the number of features, we project the data points into higher dimensions which can potentially lead to better classification results. However, a large number of features also increases the time to construct the classifier. It is therefore desirable to construct a classifier without having to include all features. Towards this goal we propose selecting features in increasing order of their computational complexity to generate a high-accuracy, low-cost classifier.

Our methods operates on the original set of features, $F = f_1, f_2, \ldots, f_n$, to produce a smaller set $\hat{F}$ by eliminating low-information and redundant features from the set using statistical measures such as the variance, $\phi(f_i)$, and the mean, $\mu(f_i)$, of the individual features. The cardinality of the feature set $\hat{F}$ is further reduced to the final set F^*. This is achieved by ordering the remaining features in increasing order of their computational complexity, $\psi(f_i)$, and adding these ordered features one by one to an initially empty set $\bar{F}$ until the accuracy of the classifier generated using $\bar{F}$, given by $K(\bar{F})$, is greater than a specified accuracy, A. The final set of features in $\bar{F}$ gives the set F^*. The individual steps are given below;

```
1. Compute $\phi(f_i)$ and $\mu(f_i)$ for all $f_i \in F$.
2. Eliminate $f_i$ if $\phi(f_i) \leq \alpha$.
3. For all $\tilde{F} \subset F$, where $\phi(f^i) = \phi(f^j)$ and $\mu(f^i) = \mu(f^j)$; for all pairs $f_i, f_j \in \tilde{F}$,
     Eliminate all but one feature $f_k$ from $\tilde{F}$, such that $\psi(f_k) = min(\psi(f_i))$, for all
        $f_i \in \tilde{F}$. Use tie breaking if necessary.
4. After completion of Step 3 we have the reduced set $\hat{F}$.
5. Order $\hat{F}$ such that if $\psi(\hat{f}_i) \leq \psi(\hat{f}_j)$, then $i < j$where $\hat{f}_i, \hat{f}_j \in \hat{F}$.
6. Set $\bar{F}$ to an empty set and $i = 0$.
7. While $K(\bar{F}) \leq A$
     $\bar{F} = \hat{f}_i \cup \bar{F}$.
     $i = i + 1$.
8. Set $F^* = \bar{F}$
```

4 Empirical Results

In this section we empirically evaluate the effectiveness of our feature selection scheme for classifier construction. Our experiments are based on a collection of 50 symmetric matrices[1], obtained from the University of Florida Sparse Matrix Collection [5]. We use a suite of linear solvers from the Portable Extensible Toolkit for Scientific Computing (PETSc) [1]. We select a combination of ten Krylov iterators and seven preconditioners and one instance where no preconditioner is used, making a total of 80 linear solvers. The Krylov methods used are: BiConjugate Gradient Squared (BCGS), Conjugate Gradient (CG), Conjugate Gradient Squared (CGS), Chebychev, Generalized Minimal Residual (GMRES) with restart values of 5, 30, and 60, Minimal Residual (MINRES), Richardson, and Transpose-Free Quasi-Minimal Residual (TFQMR).

Each Krylov method is used with the set of the following preconditioners: Incomplete Cholesky factorization (ICC), with level of fill 0, Incomplete LU Factorization (ILU) with levels of fill 0, 1, 2, and 3, Jacobi, and Successive Over Relaxation (SOR). More details about the Krylov iterators and the preconditioners can be found in [1, 20]. The total number of entries in the dataset of matrices and associated solvers is 3969 (a small percentage of entries were discarded where the simulation failed to terminate).

We use Anamod [9], developed as part of the SALSA [6] package, to calculate the properties of the matrices. Each matrix is associated with 57 features, reflecting the sparsity, variations within the matrix structure, the norm, and the spectral properties. Table 1 enumerates some of the important features, the complete list of features can be found in [8]. The complexity of calculating the features is given by $O(n)$, $O(nnz)$ and, $O(ls)$, i.e., proportional to the time taken to solve the linear system.

Our database of matrix features and solvers is classified into two groups. If for a linear system-solver pair the solution converges in 1500 iterations, the entry is labeled as group 1, otherwise it is labeled as group 0. In our dataset, out of the 3969 entries, 37.2% converged according to the solution criteria, which required the residual norm to be at least 10^{-3}. We constructed classifiers based on the learning techniques described in Section 2. We used the Weka [21] toolkit to implement these methods on a 10-fold cross validation of the dataset.

4.1 Analysis of Results

Features Set Reduction: We calculate the variance of the features in the dataset and eliminate all those whose variance is lower than 10^2. The graph of the feature variance is given in Figure 1. Out of the 57 properties, 20 have variance below this threshold. We also observe that the range of accuracy is higher when the classifiers are generated from features in the high variance range. For example, the maximum accuracy of a classifier generated using SVM with one feature from the zero variance group is at

[1] The matrix IDs are: msc00726, msc01050, msc01440, nasa2146, nasa2910, bcsstk15, msc04515, crystk01, bcsstk16, s2rmq4m1, s3rmq4m1, s3rmt3m1, nd3k, msc10848, t2dah_a, bcsstk18, vibrobox, crystm02, dis120, bodyy4, bodyy5, bodyy6, t3dl_a, bcsstk36, msc23052, smt, wathen100, fdm2, jnlbrng1, torsion1, minsurfo, c-62, c-66, nasasrb, qa8fk, qa8fm, finan512, s3dkq4m2, s3dkt3m2, m_t1, x104, shipsec8, ship_003, cfd2, augustus5, engine, pwtk, lin, af_shell4, and augustus7.

Table 1. Partial set of matrix features for matrix A generated using Anamod

Feature Name	Description	Complexity
trace	$\sum_{i=1}^{n}(a(i,i))$	O(n)
trace-abs	$\sum_{i=1}^{n}(abs(a(i,i)))$	O(n)
n-dummy-rows	number of rows with only one element	O(n)
dummy-rows-kind	For all dummy rows,D, 0 if $D(i,i)=1$; 1 if $D(i,i)\neq 1$ and $D(i,i)\neq 0$; 2 if $D(i,i)=0$	O(n)
diag-zerostart	$\min(i), \forall_{j>i} a(j,j)=0$	O(n)
diag-definite	1 if $a(i,i)>0\forall_i$, 0 otherwise	O(n)
diagonal-average	$1/n\sum_{i}^{n}(abs(a(i,i))$	O(n)
diagonal-variance	variance of the diagonal average	O(n)
diagonal-sign	-10 all zero, -2 all negative, -1 nonpositive, 0 indefinite, 1 nonnegative, 2 all positive	O(n)
nrows	number of rows in the matrix	O(n)
trace-asquared (TrAsq(A))	$\sum_{i=1}^{n}\sum_{j=1}^{n}(a(i,j)*a(j,i))$	O(nnz)
commutator-normF	$\sqrt{TrAsq(AA^T-A^TA)}$	O(nnz)
norm1	$max_i\sum_{j=1}^{n}(a(i,j))$	O(nnz)
normInf	$max_j\sum_{i=1}^{n}(a(i,j))$	O(nnz)
normF	$\sqrt{TrAsq(A)}$	O(nnz)
diagonal-dominance	$min_i(a(i,i)-\sum_{j=1}^{n}(a(i,j)))$	O(nnz)
symmetry-snorm	$max_j\sum_{i=1}^{n}(a(i,j))$, if $a(i,j)=a(j,i)$	O(nnz)
symmetry-anorm	$max_j\sum_{i=1}^{n}(a(i,j))$, if $a(i,j)\neq a(j,i)$	O(nnz)
symmetry-fsnorm	$\sqrt{TrAsq(\hat{A})}$, where $\hat{A}$ is the symmetric part of A	O(nnz)
symmetry-fanorm	$\sqrt{TrAsq(\hat{A})}$, where $\hat{A}$ is the anti-symmetric part of A	O(nnz)
nnzeros	number of nonzero elements in the matrix	O(nnz)
max-nnzeros-per-row	maximum number of nonzeros per row	O(nnz)
min-nnzeros-per-row	minimum number of nonzeros per row	O(nnz)
left-bandwidth	$\max(i-j)$, for all $a(i,j)\neq 0$	O(nnz)
right-bandwidth	$\max(j-i)$, for all $a(i,j)\neq 0$	O(nnz)
row-variability	$\max_i\log_{10}\frac{\max_j\lvert a(i,j)\rvert}{\min_j\lvert a(i,j)\rvert}$	O(nnz)
col-variability	$\max_j\log_{10}\frac{\max_i\lvert a(i,j)\rvert}{\min_i\lvert a(i,j)\rvert}$	O(nnz)
n-ritz-values	number of Ritz values	O(ls)
kappa	estimated condition number	O(ls)
positive-fraction	fraction of computed eigenvalues that has positive real part	O(ls)
sigma-max	maximum singular value	O(ls)
sigma-min	minimum singular value	O(ls)
lambda-max-by-magnitude-real	real part of maximum eigenvalue by its magnitude	O(ls)
lambda-max-by-magnitude-img	imaginary part of maximum eigenvalue by its magnitude	O(ls)
lambda-min-by-magnitude-real	real part of minimum eigenvalue by its magnitude	O(ls)
lambda-min-by-magnitude-img	imaginary part of minimum eigenvalue by its magnitude	O(ls)

most 63.69%. A similar experiment with a classifier generated from one feature from above the threshold range can yield as much as 80.67% accuracy.

We observe that, among the remaining 37 features there are some groups of redundant features such as (*row-variability*, *col-variability*), (*left-bandwidth*, *right-bandwidth*), (*n-rows*, *diag-zerostart*), (*norm1, normInf, symmetry-snorm*), (*normF, symmetry-fsnorm*), (*trace, trace-abs*), etc. By eliminating all but one feature from these groups we can further reduce the number of features to 21. Compared to the elements in the feature sets

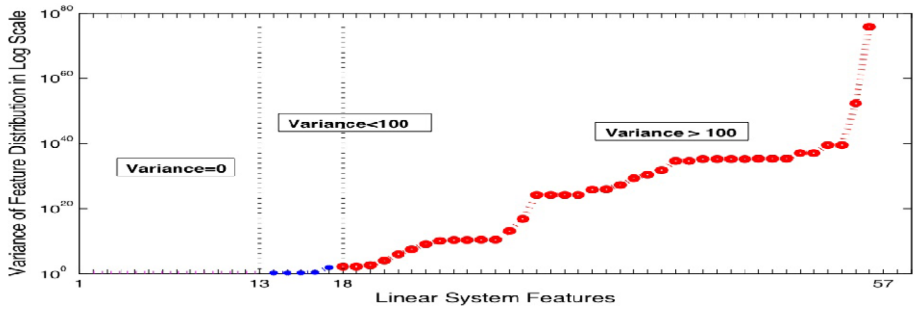


Fig. 1. Range of variance of different matrix parameters listed in Table 1

used in similar experiments, such as 66 in Xu et. al. [22], 57 in Bhowmick et. al. [3], 33 in Holloway et. al. [13] and 22 in Fuentes [11], the cardinality of the set is lower.

Feature Set Ordering: We order the features according to their computational complexity (as given in Table 1). We provide the empirical results on the classifier accuracy for two sets of orderings. The first ordering, henceforth called the *increasing order*, is based on our strategy of adding features one by one in increasing order of computational cost. The *decreasing* order denotes the case when the features are added in reverse order, i.e., the most computationally expensive feature first and so forth.

Figure 2 compares the change in accuracy as the number of features associated with the classifier is increased. The top two sub-figures represent classifiers created using NB, SVM and NN(k) with k=1 and k=10. The lower two sub-figures represent classifiers created using ADT and DS. The left hand sub-figures represent the accuracy as features are added in increasing order and the right hand sub-figures denote the accuracy as the features are added in decreasing order. The dotted lines in the figures mark the boundary between the different groups of computational complexity, i.e., $O(n)$, $O(nnz)$ and $O(ls)$. The last value in the graphs denotes the accuracy achieved by a classifier generated using all the 21 features. The point of the highest accuracy obtained is marked by an ellipse.

We observe that a set of only 9 features is sufficient to create a classifier whose accuracy is comparable the accuracy of the classifier constructed using all features. This list of features in increasing order of complexity is: trace, diagonal-variance, diagonal-average, nrows, norm1, diagonal-dominance, nnzeros, left-bandwidth, and the condition number.

We observe that for machine learning methods SVM, NB, and NN(k) the best accuracy achieved by the increasing order is comparable or even higher than the best accuracy achieved using the decreasing order. The number of features required to obtain the classifier with the best accuracy is almost equal across the increasing and decreasing orders. Since the increasing order is based on selecting features of low computational complexity, it is evident that our method will generate classifiers of lower costs.

However, in the case of the decision tree based methods (ADT and DS) the classifier does not achieve high accuracy until the condition number is added to the feature set.

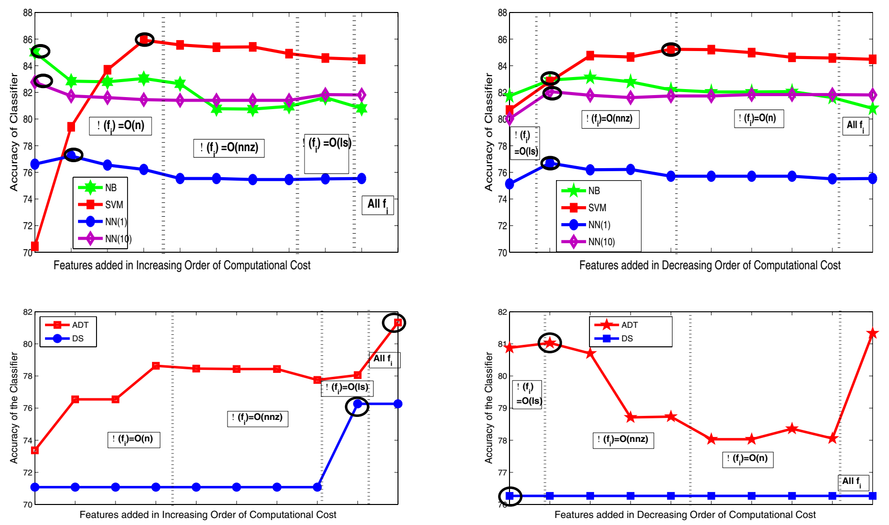


Fig. 2. Comparison of accuracy of different classifiers. Top sub-figures: Classifiers generated using NB, SVM and NN(k). Bottom sub-figures: Classifiers generated using ADT and DS. Left figures: Features added in increasing order of computational complexity. Right figures: Features added in decreasing order of computational complexity.

Table 2. Comparison of time to compute feature sets with increasing and decreasing orderings

Method	Feature Set	Computing Time (in secs)	Feature Set	Computing Time (in secs)	Speedup
	Decreasing Order		Increasing Order		
NB	(condition number left-bandwidth)	113.08	(trace)	.43	262.9
NN(1)	(condition number left-bandwidth)	113.08	(trace, diagonal-variance)	.89	127.05
NN(10)	(condition number left-bandwidth)	113.08	(trace)	.43	262.9
SVM	(condition number left-bandwidth nnzeros, diagonal-dominance)	113.95	(trace, diagonal-) variance, diagonal-average, nrows)	1.13	100.8
ADT	(condition number)	112.09	All 9	115.21	.97
DS	(condition number)	112.09	All 9	115.21	.97
Average Speedup					125.93
Geometric Mean of Speedup					30.67

Therefore decreasing order achieves the highest accuracy classifier with fewer features than the increasing order. We conjecture that this happens because the classification process depends on highly localized decisions.

Table 2 compares the time taken to compute the features for the highest accuracy classifiers (with at most the listed 9 features). Since Anamod does not give the time for calculating each individual feature, we obtain the timing results by computing the requisite feature values in MATLAB. We compute the features from a sample of the original matrix set and project the results. The values indicate that the increasing order achieves an average speedup of 125 (and a maximum of 262) compared to the decreasing order. Note that in case of ADT and DS, though all 9 features were used in the increasing order method, the feature computation time is comparable to that of the decreasing order.

We used a trial-and-error process to estimate the near-optimal feature set that generates classifiers with the highest accuracy. The highest accuracy feature sets for SVM, NB, NN(1), and NN(10) were identical with the highest accuracy ones obtained using our increasing order strategy. The near-optimal feature set for ADT and DS coincided with that obtained by the decreasing order strategy. Based on these observations, we conclude that for most machine learning methods our strategy for feature ordering achieves near-optimal classifier accuracy with low feature computation time.

5 Conclusions and Future Work

We have designed a technique for generating low-cost, high-accuracy classifiers based on ordering features in increasing order of their computational complexity. Based on the dataset and the machine learning method used, efficient feature selection can speed up the cost of building classifiers on average by a factor of 125. Most of our feature sets are identical to near-optimal ones demonstrating that our strategy achieves both low-cost and high-accuracy. Though we used our feature reduction technique on a linear system dataset, these results can easily be extended to other classification problems as well. As part of our future research we plan to study the effect of our algorithms over a variety of machine learning algorithms and datasets.

Acknowledgments

This work is a continuation of earlier research of one of the authors in application of machine learning to solver selection. We wish to thank, Victor Eijkhout, Erika Fuentes, Yoav Freund and David Keyes, the collaborators of the earlier project.

References

1. Balay, S., Buschelman, K., Gropp, W., Kaushik, D., Knepley, M., McInnes, L., Smith, B.F., Zhang, H.: PETSc Users Manual, Technical Report ANL-95/11 - Revision 2.2.1, Argonne National Laboratory (2004), `http://www.mcs.anl.gov/petsc`
2. Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H.: Templates for the Solution of Linear Systems:Building Blocks for Iterative Methods, SIAM (1994)
3. Bhowmick, S., Eijkhout, V., Freund, Y., Fuentes, E., Keyes, D.: Application of Machine Learning to the Selection of Sparse Linear Solvers,Technical Report (2007)

4. Bhowmick, S., Raghavan, P., McInnes, L., Norris, B.: Faster PDE-Based Simulations Using Robust Composite Linear Solvers. Future Generation Computer Systems 20, 373–386 (2004)
5. Davis T.: University of Florida Sparse Matrix Collection, NA Digest, 97(23) (1997), `http://www.cise.ufl.edu/research/sparse/matrices`
6. Dongarra, J., Eijkhout, V.: Self Adapting Numerical Algorithms for Next Generation Applications. International Journal of High Performance Computing Applications 17(2), 125–132 (2003)
7. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct Methods for Sparse Matrices. Clarendon Press, Oxford (1986)
8. Eijkhout V. and Fuentes E., Anamod Online Documentation, `http://www.tacc.utexas.edu/eijkhout/doc/anamod/html/`
9. Eijkhout, V., Fuentes, E.: A Proposed Standard for Numerical Metadata. ACM Trans. Math. Software (submitted)
10. Ern, A., Giovangigli, V., Keyes, D.E., Smooke, M.D.: Towards Polyalgorithmic Linear System Solvers For Nonlinear Elliptic Problems. SIAM J. Sci. Comput. 15(3), 681–703 (1994)
11. Fuentes, E.: Statistical and Machine Learning Techniques Applied to Algorithm Selection for Solving Sparse Linear Systems, Doctoral Dissertation, University of Tennessee (2007)
12. Hastie, T., Tibshirani, R., Friedman, J.H.: The Elements of Statistical Learning. Springer, Heidelberg (2001)
13. Holloway, A., Chen, T.-Y.: Neural Networks for Predicting the Behavior of Preconditioned Iterative Solvers. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part I. LNCS, vol. 4705, pp. 302–309. Springer, Heidelberg (2007)
14. Kuefler, E., Chen, T.-Y.: On Using Reinforcement Learning to Solve Sparse Linear Systems. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 955–964. Springer, Heidelberg (2008)
15. John, G.H., Kohavi, R., Pfleger, K.: Irrelevant Features and the Subset Selection Problem. In: International Conference on Machine Learning, pp. 121–129 (1994)
16. Kohavi, R., John, G.H.: Wrappers for Feature Subset Selection. Artificial Intelligence 97, 273–324 (1997)
17. Liu, H., Lei, Y.: Toward Integrating Feature Selection Algorithms for Classification and Clustering. IEEE Transactions on Knowledge and Data Engineering 17(4), 491–502 (2005)
18. McCormick, S.F. (ed.): Multigrid Methods: Theory, Applications, and Supercomputing. M. Dekker, New York (1988)
19. Nachtigal, N.M., Reddy, S.C., Trefethen, L.N.: How Fast Are Nonsymmetric Matrix Iterations? SIAM J. Matrix Anal. Appl. 13(3), 778–795 (1992)
20. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing Company (1995)
21. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco (2005)
22. Xu, S., Zhang, J.: A Data Mining Approach to Matrix Preconditioning Problem. In: Proceedings of the Eighth Workshop on Mining Scientific and Engineering Datasets (MSD 2005) (2005)

Testing Line Search Techniques for Finite Element Discretizations for Unsaturated Flow

Fred T. Tracy

Information Technology Laboratory
Engineer Research and Development Center (ERDC)
3909 Halls Ferry Road
Vicksburg, MS USA 39180
Fred.T.Tracy@usace.army.mil

Abstract. Unsaturated flow in porous media is often modeled using the finite element (FE) method. When employing an implicit version of the computational equations resulting from the FE discretization, a nonlinear system of equations is generated that must then be solved by techniques such as Newton's method. This paper reveals results of the effectiveness of three line search techniques when employed within the framework of the approximate Newton method. The methods of a bisection line search, a quadratic variation of the norm of the residual line search, and a relaxation technique are considered, all in the context of a parallel computing environment.

1 Introduction

Unsaturated flow in porous media creates special difficulties for any computational model, including the finite element method [4]. Often, an implicit method is used to prevent the need for extremely small time-steps. When this technique is employed, a significantly difficult system of nonlinear equations is generated from the discretization, often having millions of unknowns. A common solution to this system of nonlinear equations is to use the approximate Newton method [5]. Often, however, the computed change in the unknown variable (total head in the case of flow in porous media with constant density of the water) is too much, so a globalization technique of some kind [6] must be employed. Globalization can involve both line search algorithms and trust region methods. As it is not always practical to consider all points inside a trust region to get the optimum point, ways to limit the search have been developed. One such idea is the dog-leg method [7]. However, the scope of this work is to consider only line search methods. The purpose of this paper is to show results of using the three line search techniques of bisection, a quadratic variation of the norm of the residual, and a relaxation technique on a test problem that has proven especially difficult to solve. Results were obtained from a research version of a three-dimensional (3-D) finite element (FE) groundwater program developed at ERDC and running on the Cray XT3 using 16 cores.

2 Governing Equations

The version of Richards' equation used in the finite element code is

$$\nabla \cdot (k_r \mathbf{K}_s \cdot \nabla \phi) = \eta \frac{\partial S}{\partial t} + S S_s \frac{\partial \phi}{\partial t} \ , \tag{1}$$

with

$$\phi = h + z \ , \tag{2}$$

where k_r is the relative hydraulic conductivity, $\mathbf{K}_s$ is the saturated hydraulic conductivity tensor, ϕ is the total head, η is the porosity, S is the saturation, S_s is the specific storage, h is the pressure head, and z is the z coordinate. k_r and S are functions of h, thus creating the strong nonlinearity.

3 Finite Element Solution

The following equation represents the completely implicit Euler discretization of the standard continuous Galerkin technique used in the study:

$$\frac{1}{\Delta t} \mathbf{M}\left(\mathbf{u}^{n+1}\right) \cdot \left[\mathbf{u}^{n+1} - \mathbf{u}^{n}\right] + \mathbf{K}\left(\mathbf{u}^{n+1}\right) \cdot \mathbf{u}^{n+1} = \mathbf{q}\left(\mathbf{u}^{n+1}\right) \ , \tag{3}$$

where Δt is the time-step size; n is the time-step number; $\mathbf{M}$ is the mass matrix as a function of $\mathbf{u}$; $\mathbf{u}^{n+1}$ is the vector of unknown total head at the nodes at time-step, $n+1$; and $\mathbf{q}$ is the vector of known flow type terms at the nodes as a function of $\mathbf{u}$. Since $\mathbf{M}$, $\mathbf{K}$, and $\mathbf{q}$ are functions of the unknown vector, $\mathbf{u}^{n+1}$, the nonlinearity is evident. The residual at the nodes as a function of $\mathbf{u}$ is therefore

$$\mathbf{r}(\mathbf{u}) = \mathbf{q}(\mathbf{u}) - \frac{1}{\Delta t} \mathbf{M}(\mathbf{u}) \cdot [\mathbf{u} - \mathbf{u}^{n}] - \mathbf{K}(\mathbf{u}) \cdot \mathbf{u} \ . \tag{4}$$

Using, (4), column j of the approximate Jacobian matrix can now be determined by

$$\mathbf{J}_j^{n+1,k+1} = \frac{\mathbf{r}\left(\mathbf{u}^{n+1,k} + \Delta u_j \hat{\mathbf{e}}_j\right) - \mathbf{r}\left(\mathbf{u}^{n+1,k} - \Delta u_j \hat{\mathbf{e}}_j\right)}{2 \Delta u_j} \ , \tag{5}$$

where $\mathbf{u}^{n+1,k}$ is the value of $\mathbf{u}^{n+1}$ at the k^{th} nonlinear iteration, Δu_j is a small change in the j^{th} element of the vector, $\mathbf{u}^{n+1,k}$, and $\hat{\mathbf{e}}_j$ is a vector containing all zeroes except with a one in the j^{th} element. The linear system,

$$\mathbf{J}^{n+1,k+1} \cdot \delta \mathbf{u}^{n+1,k+1} = -\mathbf{r}\left(\mathbf{u}^{n+1,k}\right) \ , \tag{6}$$

can now be solved for the change in $\mathbf{u}$, and the new value of $\mathbf{u}^{n+1}$ at nonlinear iteration, $k+1$, now at least temporarily becomes

$$\mathbf{u}^{n+1,k+1} = \mathbf{u}^{n+1,k} + \delta \mathbf{u}^{n+1,k+1} \ . \tag{7}$$

4 Line Search Techniques

If $\|\delta \mathbf{u}\|_\infty$ becomes sufficiently small, the nonlinear iteration has converged. Otherwise, the process must be continued in some way. Three techniques were considered in this study, and they will now be briefly described.

4.1 Relaxation

Here, a parameter, β, where $0 < \beta_{\text{min}} \leq \beta \leq \beta_{\text{max}} \leq 1$, is used to reduce the change in $\mathbf{u}$ from one nonlinear iteration to the next. That is,

$$\mathbf{u}^{n+1,k+1} = \mathbf{u}^{n+1,k} + \beta_{k+1}\delta \mathbf{u}^{n+1,k+1} \ , \tag{8}$$

where β_{k+1} is the value of β at nonlinear iteration, $k+1$. For this option, only one value of β is considered in a given nonlinear iteration. Various values of β have been considered [2]. In this study, β was started at β_{init} and increased by β_{add} if $\|\delta \mathbf{u}^{n+1,k+1}\|_\infty \leq \|\delta \mathbf{u}^{n+1,k}\|_\infty$ (that is, $\beta_{k+2} = \beta_{k+1} + \beta_{\text{add}}$) and multiplied by β_{reduce} if $\|\delta \mathbf{u}^{n+1,k+1}\|_\infty > \|\delta \mathbf{u}^{n+1,k}\|_\infty$ (that is, $\beta_{k+2} = \beta_{\text{reduce}}\beta_{k+1}$). Care was also taken not to go smaller than β_{min} or larger than β_{max}. Values that were found to work well are $\beta_{\text{min}} = 0.1$, $\beta_{\text{max}} = 1.0$, $\beta_{\text{add}} = 0.005$, and $\beta_{\text{reduce}} = 0.677$. β_{init} varies depending on how aggressive the user chooses to be. β_{init} was set to 0.2 in this study.

4.2 Bisection Line Search

This technique is a line search where β_{k+1} is successively reduced during the given nonlinear iteration, $k+1$. First, using the notation, $\mathbf{r}\left(\mathbf{u}^{n+1,k+1}\right) = \mathbf{r}^{n+1,k+1}$, $\|\mathbf{r}^{n+1,k+1}\|_2$ is computed from (8) and (4) with β_{k+1} initially set to 1. If $\|\mathbf{r}^{n+1,k+1}\|_2$ is greater than $\|\mathbf{r}^{n+1,k}\|_2$, then β_{k+1} is reduced by one-half, and (8) and (4) are again used to compute the residual. If the norm of this new residual is smaller than $\|\mathbf{r}^{n+1,k}\|_2$, the process ends. Otherwise, β_{k+1} is again cut by one-half, and the above algorithm is repeated. After a relatively small number of bisections (maximum of 10), the attempt at a line search for this nonlinear iteration is discontinued, and the next time-step is started.

4.3 Quadratic Variation of the Norm of the Residual Line Search

Using the definitions,

$$\begin{aligned} q_1 &= \left\|\mathbf{r}\left(\mathbf{u}^{n+1,k}\right)\right\|_2 = \left\|\mathbf{r}^{n+1,k}\right\|_2 \ , \\ q_2 &= \left\|\mathbf{r}\left(\mathbf{u}^{n+1,k} + \frac{1}{2}\delta \mathbf{u}^{n+1,k+1}\right)\right\|_2 \ , \\ q_3 &= \left\|\mathbf{r}\left(\mathbf{u}^{n+1,k} + \delta \mathbf{u}^{n+1,k+1}\right)\right\|_2 \ , \end{aligned} \tag{9}$$

the norm of the residual is assumed to vary quadratically as

$$q = (2q_1 - 4q_2 + 2q_3)\,\zeta^2 - (3q_1 - 4q_2 + q_3)\,\zeta + q_1 \ , \tag{10}$$

when $0 \leq \zeta \leq 1$. Also, $\frac{dq}{d\zeta} = 0$ at

$$\zeta_0 = \frac{3q_1 - 4q_2 + q_3}{4\left(q_1 - 2q_2 + q_3\right)} \quad . \tag{11}$$

Now if $\frac{d^2q}{d\zeta^2} > 0$ and $0 < \zeta_0 < 1$, then $\beta_{k+1} = \zeta_0$ and

$$\mathbf{u}^{n+1,k+1} = \mathbf{u}^{n+1,k} + \zeta_0 \delta \mathbf{u}^{n+1,k+1} \quad . \tag{12}$$

However, if this fails but $q_3 < q_1$, then $\beta_{k+1} = 1$ and

$$\mathbf{u}^{n+1,k+1} = \mathbf{u}^{n+1,k} + \delta \mathbf{u}^{n+1,k+1} \tag{13}$$

is used. Otherwise, the $(k+1)^{\text{th}}$ nonlinear iteration is advanced one-fourth of the way with $\beta_{k+1} = 1/4$ and

$$\mathbf{u}^{n+1,k+1} = \mathbf{u}^{n+1,k} + \frac{1}{4} \delta \mathbf{u}^{n+1,k+1} \quad . \tag{14}$$

5 Test Problem with Analytical Solutions

The problem consists of a rectangular box-shaped soil sample with dimensions, $a \times b \times L$, that is initially dry until water is poured at the top of the sample (see the vertical cross-sectional view in Fig. 1). This 3-D problem has analytical solutions [9,10], and it can be made arbitrarily difficult by changing a parameter, α. In this particular test, b is made small, so the two-dimensional (2-D) version of the boundary conditions and analytical solution is used. The initial condition is $\phi = h_d + z$, where h_d is the pressure head of the soil when it is very dry. The boundary condition on the sides and bottom are the same as the initial condition, and the boundary condition on the top is

$$\phi = \frac{1}{\alpha} \ln \left[e^{\alpha h_d} + \left(1 - e^{\alpha h_d}\right) \sin \frac{\pi x}{a} \right] + L \quad , \tag{15}$$

where x is the x coordinate. Also, k_r is modeled by the quasi-linear equation [1,8],

$$k_r = e^{\alpha h} \quad , \tag{16}$$

and S varies linearly with k_r [3,11] as

$$\frac{S - S_d}{1 - S_d} = k_r \quad , \tag{17}$$

where S_d is the saturation when the soil is very dry.

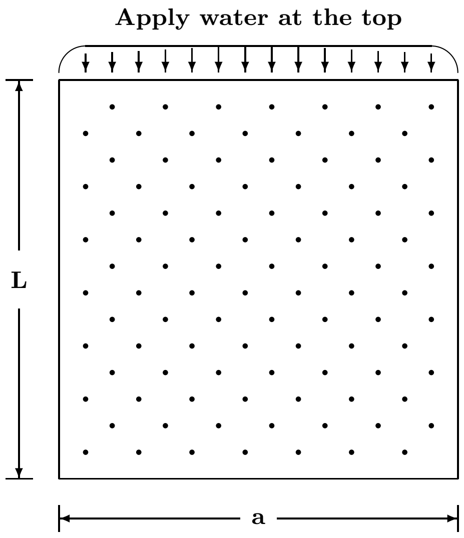


Fig. 1. Vertical cross-section of the test problem

5.1 Computational Details

The above described problem was run with $a = 50$ cm, $b = 0.25$ cm, $L = 50$ cm, $\mathbf{K}_s$ = a diagonal matrix with each diagonal element being 0.1 cm day^{-1}, $h_d = -50$ cm, $\eta = 0.45$, $S_d = 1/3$, and $S_s = 0$. The 2-D cross-section in Fig. 1 was first divided into a grid of 200 × 1 × 200 cells with $\Delta x = \Delta y = \Delta z = 0.25$ cm, and then each cell was divided into 6 tetrahedral elements, which is what the 3-D FE groundwater program requires. Data sets for relative hydraulic conductivity given in (16) and saturation modeled by (17) were then computed using 1001 values of pressure head and provided to the FE program input file. Inside the program, these curves are treated as piecewise linear.

The linear solver that was used is BiCG-Stab. Before the linear system of equations, $\mathbf{Ax} = \mathbf{b}$, as given in (6) is solved, it is normalized by

$$(\mathbf{FAF})\left(\mathbf{F}^{-1}\mathbf{x}\right) = \mathbf{Fb} \tag{18}$$

or

$$\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}} \tag{19}$$

where $\mathbf{F}$ is a diagonal matrix whose i^{th} term is

$$\mathbf{F}_{ii} = \frac{1}{\sqrt{\max\left(\max_j |a_{ij}|, \epsilon\right)}} \tag{20}$$

where a_{ij} is is the $(ij)^{\text{th}}$ component of $\mathbf{A}$, and ϵ is a small number. After the linear solve is completed, the final solution is computed by

$$\mathbf{x} = \mathbf{F}\hat{\mathbf{x}} \tag{21}$$

Since the equations are normalized with most of the diagonal terms of $\mathbf{A}$ now being 1, no further preconditioning is done in the BiCG-Stab solver.

5.2 Line Search Test Results

Table 1 shows nonlinear count results and timings for the first time-step for different values of α, Δt, and nonlinear iteration types when using a rather strong nonlinear convergence criterion of 10^{-5} for $\|\delta \mathbf{u}^{n+1,k+1}\|_\infty$. Each nonlinear iteration requires at least one linear solve of (6) and a computation of $\|\delta \mathbf{u}^{n+1,k+1}\|_\infty$. The bisection line search requires repeated calculation of the residual norm, $\|\mathbf{r}(\mathbf{u})\|_2$, which is computationally equivalent to computing the norm of the right-hand side of (6). This is an element-by-element computation of (4). The quadratic line search requires the computation of $\|\mathbf{r}(\mathbf{u})\|_2$ at the middle and end of the Newton step. The relaxation method does not require any extra computations of (4).

Each linear iteration in BiCG-Stab is dominated by two matrix-vector multiplications of the type, $\hat{\mathbf{A}}\mathbf{v}$, where $\mathbf{v}$ represents utility temporary vectors. Both $\hat{\mathbf{A}}$ and $\mathbf{v}$ are distributed over the parallel cores, thus requiring ghost node updating at strategic times using MPI. The linear solve is typically the dominant computation.

The running times given in Table 1 do not precisely delineate the impact of the different factors. To get at least a clearer understanding of these timing results, the number of linear iterations in BiCG-Stab for the first five nonlinear iterations

Table 1. Nonlinear iteration count and time in seconds for the first time-step for the three line search types, two values of α, and three values of Δt

$\alpha\,(\mathrm{cm}^{-1})$	Δt (day)	Line search type	Nonlinear iteration count	Time (sec)
0.05	0.1	Bisection	7	3.27
0.05	0.1	Quadratic	14	5.18
0.05	0.1	Relaxation	42	12.20
0.05	0.01	Bisection	8	2.77
0.05	0.01	Quadratic	15	4.21
0.05	0.01	Relaxation	45	8.37
0.05	0.001	Bisection	5	2.24
0.05	0.001	Quadratic	12	3.33
0.05	0.001	Relaxation	41	6.81
0.2	0.1	Bisection	23	5.58
0.2	0.1	Quadratic	29	7.97
0.2	0.1	Relaxation	168	27.89
0.2	0.01	Bisection	12	3.51
0.2	0.01	Quadratic	18	4.73
0.2	0.01	Relaxation	52	9.08
0.2	0.001	Bisection	∞	-
0.2	0.001	Quadratic	17	4.27
0.2	0.001	Relaxation	46	7.83

Table 2. Linear iterations required for the first five nonlinear iterations for $\alpha = 0.05\ \mathrm{cm}^{-1}$, the bisection method, and three values of Δt

Δt (day)	#1	#2	#3	#4	#5
0.1	63	44	50	30	3
0.01	24	15	13	10	6
0.001	7	4	3	1	1

of the first time-step are tabulated in Table 2 for $\alpha = 0.05\ \mathrm{cm}^{-1}$, the bisection method, and the three valuers of Δt. These results show that the number of linear iterations per nonlinear time-step is significantly reduced when the time-step size is reduced, and this helps explain the reduction in the running times in Table 1. For example, the reduction of running time from 3.27 sec to 2.77 sec while the number of nonlinear iterations increased from 7 to 8 as the time-step size is changed from 0.1 day to 0.01 day is explained by the linear iterations going from 63 to 24, 44 to 15, 50 to 13, 30 to 10, and 3 to 6, respectively.

6 Conclusions and Further Research

Conclusions that can be drawn from the above results are as follows:

1. As α is increased, the number of nonlinear iterations for all three methods increase.
2. The bisection method almost always was the most efficient. However, it stalled at an unpredictable time. It stalled because 10 unsuccessful bisections generated a very small change in total head. This failure was repeated indefinitely.
3. The relaxation method almost always was the least efficient.
4. Significant improvement in running time could potentially be achieved by implementing a more sophisticated linear solver. This is an area of future work.
5. Reducing the time-step size typically reduced both the number of nonlinear iterations and the number of linear iterations per nonlinear iteration.
6. More sophisticated nonlinear solvers/globilization techniques should be investigated to improve robustness. The one nonlinear failure reported in this paper has been observed in solving other difficult problems. Further research is needed.

Acknowledgment

This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program.

References

1. Gardner, W.: Some steady-state solutions of the unsaturated moisture flow equation with application to evaporation from a water table. Soil. Sci. 85, 228–232 (1958)
2. Gill, P., Murray, W., Wright, M.: Practical Optimization, p. 319. Academic Press, London (1981)
3. Irmay, S.: On the hydraulic conductivity of unsaturated soils. Econ. Transit., AGU 35 (1954)
4. Istok, J.: Groundwater Modeling by the Finite Element Method. AGU (1989)
5. Kelley, C.: Solving Nonlinear Equations with Newton's Method. SIAM (2003)
6. Pawlowski, R., Shadid, J., Simonis, J., Walker, H.: Globalization techniques for Newton-Krylov methods and applications to the fully coupled solution of the Navier-Stokes equations. SIAM Review 48, 700–721 (2006)
7. Pawlowski, R., Simonis, J., Walker, H., Shadid, J.: Inexact Newton dogleg methods. SIAM J. Numer. Anal. 46, 2112–2132 (2008)
8. Seǵol, G.: Classic Groundwater Simulations, p. 352. Prentice Hall, Englewood Cliffs (1994)
9. Tracy, F.: Clean two- and three-dimensional analytical solutions of Richards equation for testing numerical solvers. Water Resours. Res. 42, W08503 (2006)
10. Tracy, F.: Three-dimensional analytical solutions of Richards' equation for a box-shaped soil sample with piecewise-constant head boundary conditions on the top. J. Hyd. 336, 391–400 (2007)
11. Warrick, A.: Soil Water Dynamics, pp. 247–258. Oxford University Press, Oxford (2003)

Non-splitting Tridiagonalization of Complex Symmetric Matrices

W.N. Gansterer[1], A.R. Gruber[2], and C. Pacher[3]

[1] University of Vienna, Research Lab Computational Technologies and Applications
wilfried.gansterer@univie.ac.at
[2] University of Vienna, Institute for Theoretical Chemistry
agruber@tbi.univie.ac.at
[3] Austrian Research Centers GmbH-ARC, Department of Safety & Security
christoph.pacher@arcs.ac.at

Abstract. A non-splitting method for tridiagonalizing *complex symmetric* (*non-Hermitian*) matrices is developed and analyzed. The main objective is to exploit the purely structural symmetry in terms of runtime performance. Based on the analytical derivation of the method, Fortran implementations of a blocked variant are developed and extensively evaluated experimentally. In particular, it is illustrated that a straightforward implementation based on the analytical derivation exhibits deficiencies in terms of numerical properties. Nevertheless, it is also shown that the blocked non-splitting method shows very promising results in terms of runtime performance. On average, a speed-up of more than three is achieved over competing methods. Although more work is needed to improve the numerical properties of the non-splitting tridiagonalization method, the runtime performance achieved with this non-unitary tridiagonalization process is very encouraging and indicates important research directions for this class of eigenproblems.

Keywords: Tridiagonalization, complex symmetric eigenvalue problems, complex symmetric reflector.

1 Introduction

We discuss an algorithm for tridiagonalizing a *complex symmetric* (*non Hermitian*) matrix $C \in \mathbb{C}^{n\times n}$. This task is a central component in reduction methods for solving the complex symmetric eigenvalue problem (EVP)

$$Cx = \lambda x \quad \text{with} \quad C \in \mathbb{C}^{n\times n},\ C = C^\top. \tag{1}$$

Problems of this type are a special case of general non Hermitian complex eigenproblems. Although they do not occur as frequently in practice as real symmetric or complex Hermitian problems, there are many important applications where they arise [1]. An example is the numerical solution of Maxwell's equations with complex material coefficients (accounting for losses) and/or certain absorbing boundary conditions used in the simulation of optoelectronic devices [2].

Especially for large n, it is pivotal to exploit the (non Hermitian) symmetry present in problem (1) in order to be able to efficiently solve such problems.

Hardly any high quality software is available for complex symmetric eigenproblems. Only QMRPACK [3] contains an implementation of a complex symmetric Lanczos algorithm. General purpose state-of-the-art software libraries for dense numerical linear algebra computations, such as the BLAS [4] and LAPACK [5] for sequential computations, or the parallel packages SCALAPACK [6] and PLAPACK [7], contain very few computational routines which are capable of specifically exploiting complex symmetry. No such routine for the complex symmetric eigenvalue problem is currently available in these state-of-the-art software packages. Consequently, the currently most common strategy for solving problems (1) is to ignore their special properties and to solve them with the technology available for general non Hermitian problems (for example, using the routine `LAPACK/zgeev`): The complex symmetric matrix C is first reduced to Hessenberg form using unitary transformations, from which eigenvalues and eigenvectors are computed by applying standard methods for unsymmetric matrices. This strategy has obvious disadvantages in terms of computational effort and in terms of storage requirements.

The main objective of this paper is to investigate a *non-splitting* approach for tridiagonalizing C and to compare it on the one hand to standard methods for general non Hermitian eigenproblems, and on the other hand to the splitting tridiagonalization method for complex symmetric matrices discussed earlier [2].

Related Work. Various projection methods for solving complex symmetric EVPs have been proposed, for example, based on modifications of the non-Hermitian Lanczos method [3,8,9], on subspace iteration [10], or on variants of the Jacobi-Davidson method [11].

For dense matrices and/or if large parts of the spectrum are to be computed, transformation methods (based on tridiagonalization) can be more efficient [12]. For these methods, the tridiagonalization step tends to be a dominating part in terms of arithmetic complexity and usually also computation time. Earlier attempts were based on modifying the conventional Householder-based tridiagonalization for real symmetric or complex Hermitian matrices such that symmetry is preserved for complex symmetric problems [13]. In [13], the Householder vector is normalized using a quasi-inner product, which has several implications: The normalization factor can become a negative or a complex number, and the resulting transformation matrices are not unitary. The idea of tridiagonalizing real and imaginary part of C separately, connected by complex orthogonal transformations (called *splitting method* in the following) for tridiagonalizing a complex symmetric matrix has been investigated in [14,2].

For computing eigenvalues and eigenvectors of the resulting tridiagonal complex symmetric problem, modifications of the QR algorithm [15,16] and the routines `cmtql1` and `inverm` [17] have been used.

In this paper, we investigate *non-splitting* methods as an alternative approach for tridiagonalizing a complex symmetric matrix C. The basic concept is very similar to the one mentioned in [13]. However, in [13] neither a discussion of numerical properties (observed deficiencies in accuracy, special techniques for overcoming them, etc.) nor a quantitative performance evaluation of this approach is given. We provide a comparison of this approach to the splitting method in terms of numerical properties and runtime performance.

Synopsis. In Section 2 we review general properties of complex symmetric matrices and we summarize the splitting method introduced earlier. In Section 3, we derive a non-splitting tridiagonalization method for complex symmetric matrices and relate it to generalizations of unitary Householder reflectors. In Section 4, an experimental evaluation of this non-splitting method and a comparison to other approaches is summarized, and Section 5 contains conclusions and future work.

2 Background

Mathematically speaking, structural symmetry is not a very distinctive feature of complex matrices, since *every* matrix $A \in \mathbb{C}^{n\times n}$ is similar to a complex symmetric matrix [1]. In contrast to a real symmetric matrix, a complex symmetric matrix A is not necessarily normal and not necessarily diagonalizable [1]. Nevertheless, the purely algebraic symmetry is of great interest for the development of space- and time-efficient algorithms. Obviously, half of the information in a complex symmetric matrix is redundant, and efficient algorithms should be able to take advantage of this fact in terms of memory requirements as well as in terms of computational effort.

2.1 Complex Orthogonal Transformations

As a basic guideline, there are two conditions for any transformation used in the tridiagonalization process of C: (i) It has to be a *similarity transformation* in order to preserve the spectrum of C, and (ii) it has to be *symmetry-preserving* in order to exploit the structural symmetry. Consequently, given a transformation matrix $G \in \mathbb{C}^{n\times n}$, each transformation needs to be of the form GCG^{-1} in order to satisfy the first condition and of the form $GCG^\top$ in order to satisfay the second condition. In summary, the basic transformation matrices used need to satisfy

$$GG^\top = I, \tag{2}$$

which defines a *complex orthogonal* transformation [1] (COT) G. Note that G is *not* unitary and in general $\|G\|_2 > 1$. Consequently, the application of complex orthogonal transformations G potentially involves compromising numerical stability. In order to bound the numerical errors in transformation processes using complex orthogonal matrices, their norms have to be monitored, and, if possible, kept below some properly chosen threshold.

2.2 The Splitting Method

The splitting method, which has been introduced in [14], is based on separating the tridiagonalization of the real part R of C from the tridiagonalization of the imaginary part S of C (R and S are both real symmetric matrices) as much as possible. For example, the part below the subdiagonal of a column of R (say, of length k) can be eliminated using a real orthogonal transformation matrix Q_{R}. After that, a $k-1$ part of the corresponding column of S can be eliminated without causing any fill-in in R using another real orthogonal matrix Q_{I}. Both of these operations are performed in real arithmetic, and both transformation matrices have norm one. Eventually, a single nonzero element below the subdiagonal in S remains to be eliminated. This operation has to be performed in complex arithmetic, using a 2×2 complex orthogonal transformation matrix, whose norm cannot be bounded a priori.

In [2], we have investigated numerical properties and runtime performance of the splitting method for tridiagonalizing C, and we have pointed out some algorithmic variants which have not been mentioned in [14]. Our experimental results showed that the splitting method can achieve good numerical accuracy, but the runtime performance achieved was often not better than the one achieved with the routine `zgeev` from LAPACK [5].

3 Non-splitting Tridiagonalization Methods

Non-splitting methods for tridiagonalizing C are characterized by the fact that, in contrast to the splitting method, they do not split up C into its real and imaginary parts but operate in complex arithmetic on the complex symmetric matrix as a whole. The method investigated in this paper can be derived from an old method proposed by La Budde [18] for tridiagonalizing any real *unsymmetric* matrix. In the following, we first briefly review La Budde's method and then show how it can be modified to tridiagonalize a complex symmetric matrix.

3.1 La Budde's Method

In order to eliminate a column vector $c \in \mathbb{R}^{n-j}$ below the subdiagonal and a row vector $b \in \mathbb{R}^{n-j}$ right of the superdiagonal in an unsymmetric real $n \times n$ matrix, La Budde [18] determined elimination vectors $x, y \in \mathbb{R}^{n-j}$, such that the matrices

$$M_\alpha := I_{n-j} + \alpha x y^\top, \qquad \alpha \neq 0, \tag{3}$$

$$M_\beta := I_{n-j} + \beta x y^\top, \qquad \beta \neq 0, \tag{4}$$

constitute a similarity transformation ($M_\alpha M_\beta = I_{n-j}$) and eliminate all entries except the first one in the vectors c and b:

$$M_\alpha c = \tau_1 e_1,$$
$$b^T M_\beta = \tau_2 e_1^T,$$

where $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^{n-j}$. Carrying out this process over columns $j = 1, 2, \ldots, n-2$, the given matrix can be transformed to tridiagonal form.

It turns out that La Budde's method breaks down when at some point in the process $s := b^\top c = 0$. La Budde suggested in [18] to avoid this breakdown by a proper choice of the parameters α and β in (3) and (4). However, it was pointed out in [19] that there are at least two categories of matrices where it is *not* possible to choose α and β such that recovery from $s = 0$ is possible. Remedies were proposed for these cases. Parlett [20] has pointed out that La Budde's procedure applied to Hessenberg matrices is identical to well-known transformation methods. He also showed that for Hessenberg matrices s is invariant for all permissible choices of α and β. Therefore, Hessenberg matrices form another class of matrices where recovery from breakdown is not possible by manipulating α and β.

3.2 Adaptation for Complex Symmetric Matrices

In the complex symmetric case b, c, x, y are complex vectors and $c = b$, $\alpha = \beta$, $x = y$. Therefore, $M_\alpha = M_\beta$ and $M_\alpha = M_\alpha^T$. This yields the following equations:

$$M_\alpha = I_{n-j} + \alpha x x^T, \quad \alpha \neq 0, \tag{5}$$

$$M_\alpha^2 = I_{n-j}, \tag{6}$$

$$M_\alpha c = \tau e_1. \tag{7}$$

From (5) and (6) and from the fact that in the nontrivial case $M_\alpha \neq I_{n-j}$ we can conclude $\alpha = -2/x^T x$ and thus

$$M_\alpha = I_{n-j} - \frac{2}{x^T x} x x^T. \tag{8}$$

Note that in the symmetric case M_α does in fact not depend on α any more and thus we drop the index α in the following.

Now, the vector x has to be determined so that (7) is satisfied. From (7) we obtain $\tau^2 = \tau e_1^T e_1 \tau = c^T M^T M c = c^T c$, where the last equation follows from $M^T M = MM = I$. Consequently

$$\tau = \pm\sqrt{c^T c},$$

and the resulting vector

$$\tau e_1 = \pm\sqrt{c^T c}\, e_1.$$

Thus, we have

$$Mc = \pm\sqrt{c^T c}\, e_1 \quad \Leftrightarrow \quad c - \frac{2}{x^T x}(x^T c)x = \pm\sqrt{c^T c}\, e_1,$$

which we must solve for x. Note that x can be scaled by any (complex) number a without changing M. If we scale x such that $\frac{2}{x^T x}(x^T c) = 1$ holds, then the following solutions are easily found:

$$x = c \mp \sqrt{c^T c}\, e_1.$$

Consequently, we obtain the following equations for the components of the elimination vector x:

$$x_1 = c_1 \mp \sqrt{c^T c} = c_1 - \tau, \tag{9}$$

$$x_k = c_k, \qquad k = 2, \ldots, n. \tag{10}$$

Note that this is analogous to the definition of the real Householder reflector. This shows on the one hand that in the real symmetric case La Budde's method is identical to real Householder reflectors, and on the other hand that in the complex symmetric case it leads to a direct generalization of Householder reflectors in complex arithmetic.

3.3 Numerical Properties

Since for a vector $x \in \mathbb{C}^{n-j}$, $x^\top x = 0$ is possible even if $x \neq 0$, (8) indicates that $||M|| \geq 1$ and that breakdown and numerical problems are possible. For the splitting method, various *recovery transformations* for controlling and improving numerical accuracy have already been investigated [14]. Our derivation above illustrates that the need for monitoring and improving numerical accuracy also arises in the context of non-splitting methods. The investigation and development of suitable recovery transformations in this context is topic of ongoing work. In the following section, we evaluate a straightforward implementation of the basic non-splitting method based on (9) and (10).

4 Experimental Evaluation

For our experiments, we implemented the following routines in Fortran: `zsytd2` performs an *unblocked* non-splitting tridiagonalization of a complex-symmetric matrix C, `zsytrd` performs a *blocked* non-splitting tridiagonalization of complex-symmetric C, `zunm2r2` performs an *unblocked* backtransformation of the eigenvectors of the tridiagonal matrix T to those of C, and `zunmtr2` performs a *blocked* backtransformation of the eigenvectors of the tridiagonal matrix T to those of C. The routines `compev` [17] and `inverm` [17] were used for computing eigenvalues and corresponding eigenvectors of the complex symmetric tridiagonal matrix. For comparison, the routine `LAPACK/zgeev` for general non Hermitian eigenvalue problems was used.

The codes were run on a Sun Fire v40z with 4 dual-core Opteron 875 CPUs (2.2 GHz) and 24 GB main memory. Suse Linux Enterprise Server 10, the GNU Fortran 95 compiler, LAPACK version 3.1.1 and GOTO BLAS 1.20 were used. The test matrices were created randomly.

4.1 Numerical Accuracy

Denoting with (λ_i, x_i) the eigenpairs computed by `LAPACK/zgeev`, and with $(\tilde{\lambda}_i, \tilde{x}_i)$ the eigenpairs computed by non-splitting tridiagonalization followed by

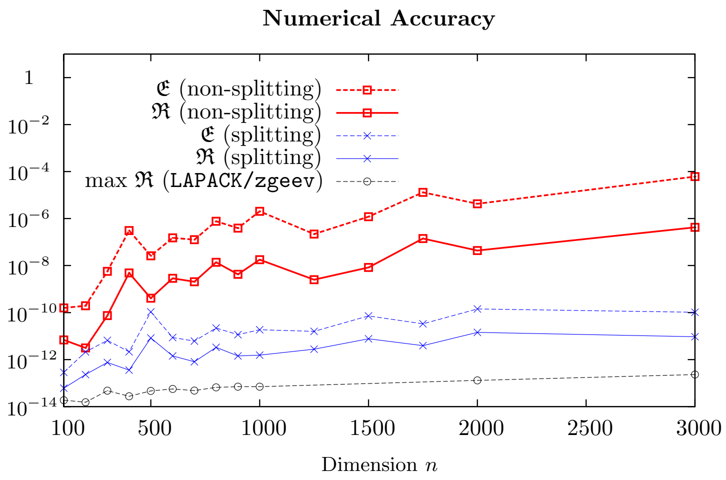


Fig. 1. Residuals and eigenvalue errors for different complex symmetric eigensolvers

`compev` and `inverm`, an average eigenvalue error $\mathfrak{E}$ and an average residual error $\mathfrak{R}$ have been computed according to

$$\mathfrak{E} := \text{average}_i \frac{|\tilde{\lambda}_i - \lambda_i|}{|\lambda_i|} \ , \quad \mathfrak{R} = \text{average}_i \frac{||(A - \tilde{\lambda}_i I_n)\tilde{x}_i||_2}{||A||_2}, \quad i \in \{1, \dots, n\} \ .$$

Fig. 1 illustrates the experimental results. It has already been shown in [2] that the error introduced by the splitting tridiagonalization (*without* the solver for the tridiagonal problem) is only about two orders of magnitudes higher than the one of `LAPACK/zgeev`. The first straightforward implementation of the non-splitting method used here looses some more orders of magnitude in accuracy compared to the splitting method. (Note that for `LAPACK/zgeev` the maximum residuals are shown, not their averages.) This indicates the need for improvements of the numerical properties of the non-splitting tridiagonalization approach, as already mentioned in Section 3.3. Corresponding efforts are work in progress.

4.2 Runtime Performance

In [2] we have already illustrated that with the Goto Blas, `LAPACK/zgeev` tends to outperform the splitting method in terms of runtime performance. Thus, in Fig. 2 we compare normalized runtimes of blocked non-splitting tridiagonalization followed by `compev`, `inverm` and blocked backtransformation (the sum of these runtimes is denoted by `nonsplit`) to normalized runtimes of `LAPACK/zgeev`. Fig. 2 illustrates the big potential of non-splitting tridiagonalization: Its runtime performance is significantly better than the one of `LAPACK/zgeev` and consequently also than the one of the splitting method. Despite a similar asymptotic behavior, the non-splitting approach achieves on average a speed-up of a little more than three over `LAPACK/zgeev`.

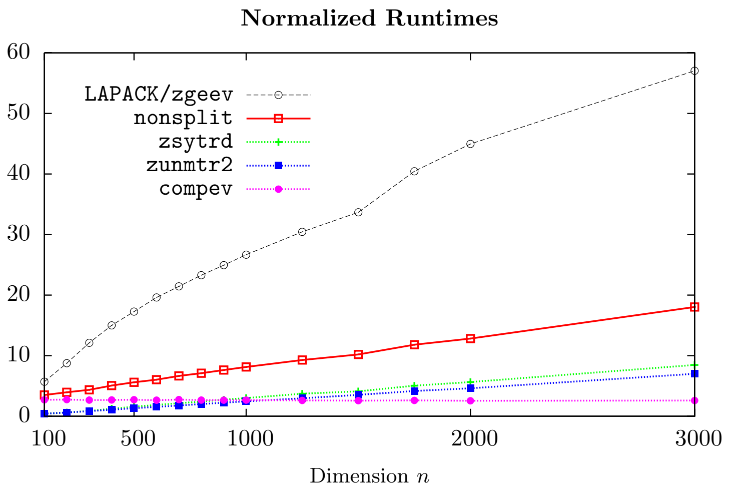


Fig. 2. Normalized runtimes ($T(n)/n^2$) of complex symmetric eigensolver (`nonsplit`) based on non-splitting tridiagonalization, of its components, and of `LAPACK/zgeev`

This underlines the importance of efforts in improving the numerical properties of the non-splitting approach. If they can be improved by some properly chosen recovery transformations in cases where the naive implementation used here suffers from numerical inaccuracies, then a central building block can be established for a very competitive method for solving dense complex symmetric eigenproblems ([1]).

5 Conclusions and Future Work

A non-splitting tridiagonalization process for complex symmetric matrices which can be derived from a method for tridiagonalizing unsymmetric matrices originally suggested in [18] has been investigated. It has been shown that this approach leads to a symmetry preserving generalization of complex Householder reflectors. The non-splitting tridiagonalization process and a complex symmetric eigensolver based on it have been analyzed in terms of numerical properties and runtime performance.

Compared to the standard Lapack routine for general non Hermitian eigenproblems, `LAPACK/zgeev`, and to the splitting method proposed and analyzed earlier [14,2], the naive straightforward implementation of the non-splitting approach exhibits a loss of numerical accuracy (measured in terms of eigenvalue error and in terms of residual error). Potential sources for this loss of accuracy are clear, and work on improving the numerical properties of the non-splitting method is in progress. In terms of runtime performance, a blocked non-splitting approach shows very promising results. In the experiments summarized in this paper, on average a speed-up of more than three could be achieved over competing methods. If progress can be made in the numerical aspects, then non-splitting methods have the potential to become a standard approach for tridiagonalizing complex symmetric matrices.

The work summarized here motivates various further research directions. As mentioned before, the improvement of the numerical properties of non-splitting tridiagonalization is of utmost importance. Once this aspect is sucessfully addressed, it should be possible to develop parallelization strategies analogously to Householder-based tridiagonalization methods for real symmetric or complex Hermitian matrices.

References

1. Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press, Cambridge (1985)
2. Gansterer, W.N., Schabauer, H., Pacher, C., Finger, N.: Tridiagonalizing complex symmetric matrices in waveguide simulations. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 945–954. Springer, Heidelberg (2008)
3. Freund, R.W., Nachtigal, N.M.: QMRPACK: a package of QMR algorithms. ACM Trans. Math. Softw. 22(1), 46–77 (1996)
4. Dongarra, J.J., Du Croz, J., Duff, I.S., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. 16, 1–17, 18–28 (1990)
5. Anderson, E., Bai, Z., Bischof, C.H., Blackford, S., Demmel, J.W., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.C.: Lapack Users Guide, 3rd edn. SIAM Press, Philadelphia (1999)
6. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J.W., Dhillon, I., Dongarra, J.J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLapack Users' Guide. SIAM Press, Philadelphia (1997)
7. van de Geijn, R.: Using PLapack: Parallel Linear Algebra Package. The MIT Press, Cambridge (1997)
8. Freund, R.W., Gutknecht, M.H., Nachtigal, N.M.: An implementation of the look-ahead Lanczos algorithm for non-hermitian matrices. SIAM J. Sci. Comput. 14(1), 137–158 (1993)
9. Cullum, J.K., Willoughby, R.A.: A practical procedure for computing eigenvalues of large sparse nonsymmetric matrices. In: Cullum, J.K., Willoughby, R.A. (eds.) Proceedings of the IBM Europe Institute Workshop on Large Scale Eigenvalue Problems, pp. 193–223. North-Holland, Amsterdam (1986)
10. Leung, A.Y.T.: Subspace iteration for complex symmetric eigenproblems. J. Sound Vibration 184(4), 627–637 (1995)
11. Arbenz, P., Hochstenbach, M.E.: A Jacobi–Davidson method for solving complex symmetric eigenvalue problems. SIAM J. Sci. Comput. 25(5), 1655–1673 (2004)
12. Bai, Z., Demmel, J., Dongarra, J.J., Ruhe, A., van der Vorst, H. (eds.): Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM Press, Philadelphia (2000)
13. Ohnami, K., Mikami, Y.: Resonance scattering in a two-dimensional non-integrable system. J. Phys. A 25, 4903–4912 (1992)
14. Bar-On, I., Ryaboy, V.: Fast diagonalization of large and dense complex symmetric matrices, with applications to quantum reaction dynamics. SIAM J. Sci. Comput. 18, 1412–1435 (1997)

15. Cullum, J.K., Willoughby, R.A.: A QL procedure for computing the eigenvalues of complex symmetric tridiagonal matrices. SIAM J. Matrix Anal. Appl. 17, 83–109 (1996)
16. Luk, F., Qiao, S.: Using complex-orthogonal transformations to diagonalize a complex symmetric matrix. In: Luk, F.T. (ed.) Advanced Signal Processing: Algorithms, Architectures, and Implementations VII, Proc. SPIE, vol. (3162), pp. 418–425 (1997)
17. Cullum, J.K., Willoughby, R.A.: Lanczos Algorithms for Large Symmetric Eigenvalue Computations, vol. 1: Theory, vol. 2: Programs. Birkhäuser, Boston (1985)
18. La Budde, C.D.: The reduction of an arbitrary real square matrix to tridiagonal form using similarity transformations. Math. Comp. 17, 433–437 (1963)
19. Wang, H.H., Gregory, R.T.: On the reduction of an arbitrary real square matrix to tridiagonal form. Math. Comp. 18, 501–505 (1964)
20. Parlett, B.N.: A note on La Budde's algorithm. Math. Comp. 18, 505–506 (1964)

Parallel MLEM on Multicore Architectures

Tilman Küstner[1], Josef Weidendorfer[1], Jasmine Schirmer[2], Tobias Klug[1], Carsten Trinitis[1], and Sybille Ziegler[2]

[1] Department of Computer Science, Technische Universität München
{tilman.kuestner,josef.weidendorfer,tobias.klug, carsten.trinitis}@cs.tum.edu

[2] Department of Nuclear Medicine, Technische Universität München
jasmine.schirmer@tum.de, s.ziegler@lrz.tu-muenchen.de

Abstract. The efficient use of multicore architectures for sparse matrix-vector multiplication (SpMV) is currently an open challenge. One algorithm which makes use of SpMV is the maximum likelihood expectation maximization (MLEM) algorithm. When using MLEM for positron emission tomography (PET) image reconstruction, one requires a particularly large matrix. We present a new storage scheme for this type of matrix which cuts the memory requirements by half, compared to the widely-used compressed sparse row format. For parallelization we combine the two partitioning techniques recursive bisection and striping. Our results show good load balancing and cache behavior. We also give speedup measurements on various modern multicore systems.

1 Introduction

In contrast to computer tomography (CT), which aims at structural imaging, positron emission tomography (PET) visualizes functional processes, by measuring the distribution of a tracer consisting of radioisotopes injected into a patients body. Clinical PET scanners for example assist in tumor diagnosis. PET research currently focuses on improving spatial resolution and sensitivity of the technique.

A PET scanner consists of fixed detectors, usually arranged in a ring around the subject to be analyzed. A positron-emitting radioisotope can be detected indirectly, as positrons annihilate with electrons, creating two 511 keV gamma photons traveling in opposite directions. When two detectors each record a photon within a certain time window, an annihilation event is assumed somewhere along the line connecting the detectors. This line is called the line of response (LOR). The number of detected events influences the quality of the measurement, while the coverage of three-dimensional space of interest (field of view, FOV) by LORs affects the achievable resolution. The resolution is usually better at the center than at the edges of the field of view. The FOV is commonly divided into a three-dimensional grid, where each grid cell is called a voxel.

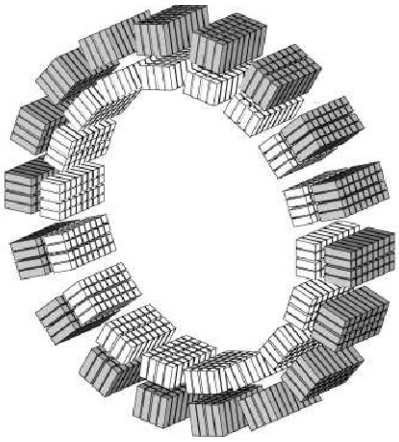


Fig. 1. Geometry of MADPET-II

The experimental small-animal scanner MADPET-II [8] was developed at the Department of Nuclear Medicine. This scanner is able to resolve the issue of poor spatial resolution by adding a second ring of detectors (see Fig. 1). This leads to a quadratic increase of measurement data, and consequently, a significant increase in computational demand for the post processing step, the 3D image reconstruction.

Several algorithms can be used for reconstructing PET images. One is filtered back-projection (FBP), which is based on an analytic solution of the Radon transform. Other algorithms are based on iterative reconstruction such as the maximum likelihood maximization (MLEM) algorithm. This algorithm usually outperforms FBP in terms of image quality, but is more computationally intensive.

The huge memory requirements of MLEM come from fixed input data, describing the geometrical and physical properties of the scanner. This data is arranged in a matrix which gives the probability of an event occurring in one voxel being recorded by a given pair of detectors. This so-called system matrix is sparse, since for voxels outside a given LOR, the detection probability is almost certainly zero. The system matrix can be measured in a physical experiment, or it can be computed from analytic models or by Monte Carlo simulation. The simulation takes a number of physical effects into account, which influence the trajectory and detection of the photons.

The MLEM algorithm is iterative meaning that it starts with an estimate of the solution, which is then corrected in every iteration step. In each step, two vector scaling operations and two sparse matrix-vector multiplications (SpMV or sometimes SpMxV) are carried out. The latter operations are known for a number of performance problems [3]. Amongst them are indirect referencing and irregular access of the source vector as well as high load on the memory subsystem created by the traversal of the matrix. We parallelize the MLEM algorithm by splitting the SpMV operations into smaller ones. As target systems, we have computer clusters using multicore processors in mind.

2 Related Work

A comprehensive overview of iterative algorithms for image reconstruction in general is given in [7]. Parallel algorithms for reconstructing both CT and PET images are described in [5]. The article also provides an overview of feasible acceleration techniques, and covers a broad range of parallel environments, from networks of workstations, to peer-to-peer and grid computing.

During the 1990s, various approaches to parallelizing MLEM have been proposed [1,2,6]. They exploit the symmetries in the scanner geometry and partition the FOV and thus the image vector in a compatible way. All proposals are limited by the computational power available in the respective years. Thus, in all the articles cited, image, measurement and system data is much smaller than in our work.

Given future systems with accelerators, another method of dealing with the huge memory requirements is computing the system matrix on the fly from a simplified analytical model. In a previous work [4] we made use of this technique, implementing the MLEM algorithm on the IBM Cell BE and using the fast accelerator cores of the Cell processor. Currently, more work is done here to test more accurate and faster analytical models [10].

3 The MLEM Algorithm

The MLEM algorithm was first proposed by Shepp and Vardi [11]. It can be viewed as an implementation of the more general expectation-maximization (EM) algorithm, applied to the problem of image reconstruction. In the following, we will give a short formal explanation of the algorithm.

The system matrix $A = (a_{ij})$ gives the probability of a photon emitted from voxel j being recorded by detector pair i. Its number of columns m equals the number of voxels, its n rows correspond to the n detector pairs or LORs. Let f denote the image vector and g the measuring vector. Disregarding all stochastic effects, we can state that the MLEM algorithm tries to approximate a solution of the set of linear equations $Af = g$.

An MLEM iteration step is given by

$$f_j^{(q+1)} = \frac{f_j^{(q)}}{\sum_{l=1}^{n} a_{lj}} \sum_{i=1}^{n} a_{ij} \frac{g_i}{\sum_{k=1}^{m} a_{ik} f_k^{(q)}} \quad . \tag{1}$$

We use the superscript q to denote the iteration number. One step can be divided into several parts. First, the forward projection (FP) is calculated:

$$h_i^{(q)} = \sum_{k=1}^{m} a_{ik} f_k^{(q)} \tag{2}$$

This equation describes a multiplication of the sparse matrix A with the image vector $f^{(q)}$. The resulting vector $h^{(q)}$ shows what the measurement would look

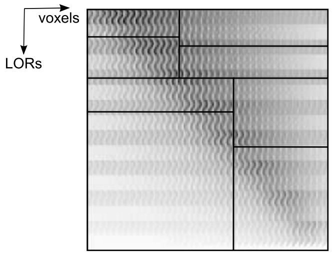


Fig. 2. Density plot and possible partitioning of matrix II

like for the approximate image vector $f^{(q)}$. Subsequently, the forward projection is compared to the actual measurement and a correction factor is derived:

$$c_j^{(q)} = \sum_{i=1}^{n} a_{ij} \frac{g_i}{h_i^{(q)}} \tag{3}$$

This step is called back projection (BP) and matches a sparse matrix-vector multiplication, this time using the transposed matrix. Finally, the vector of correction factors and an additional scaling factor are applied to the image vector (see Eq. 1). The scaling factor can of course be calculated before the iteration starts.

4 Implementation

To come up with an efficient implementation of the MLEM algorithm, a fitting storage format for the system matrix has to be created. Our two test matrices, labeled I and II, describe the sensitivity of the small animal pet scanner MADPET-II [8], which has 1152 detectors arranged in two rings (see Fig 1). This results in $n = 662976$ lines of response or matrix rows. The field of view is divided into $m = 784000$ voxels arranged in a $140 \times 140 \times 40$ grid.

The test matrices were both generated by Monte Carlo simulation, but with different parameters. Two million annihilation events per voxel were simulated. Besides the information of the LOR, the simulation also returns the absorption energy of a detection. Whether a given detection is counted for the resulting matrix depends on a chosen energy threshold. The energy threshold for matrix I was set to 400 keV, resulting in a 2.6 GB matrix, whereas the energy threshold for matrix II was set to 200 keV, resulting in a 17.6 GB matrix. Matrix I has 0.12% non-zero entries, whereas the sparsity of matrix II is 0.84%.

To provide an impression of the matrix structure we include a grayscale plot of matrix II in Fig. 2.

4.1 Matrix Storage Format

As the same number of events is simulated in every voxel, only the number of successfully detected events needs to be stored in the matrix. Even in the longest running Monte Carlo simulation, the largest number of successfull recordings was below 100, because a huge amount of generated photon pairs completely miss any detector.

We use a modified version of the compressed sparse row format (CSR) to store the matrix. Because of the low number of possible values of a matrix element, an array to explicitly store the values is not needed. Let $\tilde{a}_i$ denote the maximal entry of each matrix row. In detail, the three arrays of our matrix format are the following:

- *column* – An array whose length equals the total number of matrix entries. It contains the column numbers of the entries. The elements are sorted into sections according to their respective row number. Within the sections, the elements are sorted in ascending order of their value.
- *value* – An array of length $\sum \tilde{a}_i + 1$. It contains indices of the array *column*. The element $value[j]$ points to the column number of a matrix element in row i with a value of $j - row[i] + 1$.
- *row* – An array of length $n + 1$, containing indices of the array *value*. The element $row[i]$ points to the first non-zero element of row i.

As an example, we show how the following matrix is stored in our format.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \quad \rightarrow \quad \begin{array}{l|l} row & 0\,1\,3\,6\,6\,7 \\ \hline value & 0\,4\,6\,7\,7\,8\,9\,11 \\ \hline column & 0\,1\,2\,3\,1\,3\,2\;\;2\;\;1\,2\,3 \end{array}$$

By using this format, we can store a system matrix in about half the amount of memory that would be required for the CSR format, due to the short length of the array *value*.

It should be noted that the multiplication with the transposed matrix in the second stage of the MLEM algorithm does not raise any additional issues. The same matrix format can be used; only the type of access to the source and destination vectors is interchanged. During BP the source vector is accessed sequentially, whereas the destination vector is accessed according to the sparsity pattern of the matrix.

4.2 Parallelization

Having cluster environments in mind, we use the message passing interface (MPI) as a basis of our parallelization. On multicore architectures, MPI uses shared memory for communication. Future work will evaluate the performance of a hybrid parallelization, i.e. OpenMP within the nodes and MPI as an inter-node communication model on computer clusters.

We use recursive bisection to parallelize the SpMV operations. Blocks of equal numbers of non-zero matrix elements are created. We then assign each block to one MPI process and each process is mapped to one processor core. Using this "owner computes" approach, each core calculates a small SpMV operation. The partial results are gathered in an all-reduce operation. The blocks resulting from recursively dividing matrix II three times are displayed in Fig. 2.

If the number of columns of one block is large, it can be further subdivided into vertical stripes. In terms of strategies for cache optimization this is known as one-dimensional blocking. The stripes are processed in sequential order by the core processing the block. Creating stripes turns out to be especially beneficial on machines with small last-level CPU cache, as will be shown in section 5.1. Striping creates sections in the vector that is affected by the problem of irregular access. The sections fit into the last-level cache. This applies to the source vector of the multiplication during forward projection and the destination vector during back projection.

Given this parallelization idea, a process only needs to load its assigned block of the system matrix. Thus, the huge memory consumption of a large matrix can be distributed to the nodes of a computer cluster or the nodes of a machine with non-uniform memory access.

5 Experiments and Results

In this section we present results gained from tests on five machines with both uniform (UMA) and non-uniform memory access (NUMA). We use Intel Compiler 10.1 with the option -O3 for compiling and OpenMPI 1.3 as MPI runtime environment. In order to obtain reliable results we pin the MPI processes to CPU cores using the tool `taskset`[1]. The importance of pinning will be shown in section 5.3.

The first system consists of two Intel Xeon 5335 (Clovertown) processors at 2.6 GHz and 8 GB main memory. The Clovertown processor has four cores, with two cores sharing a 4 MB L2 cache each. See Fig. 3 (a) for the design of this UMA system that will be referred to as Clovertown.

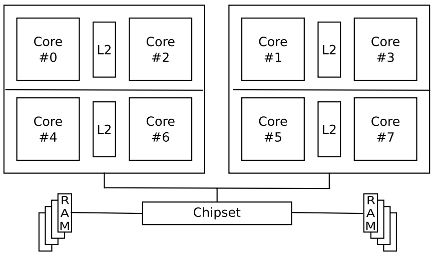
The second system is equipped with two AMD Opteron 2352 (Barcelona) processors at 2.1 GHz and 16 GB main memory. Each CPU has four cores, each of which has a private L2 cache. All cores on a chip share a 2 MB L3 cache. This NUMA machine will be referred to as Barcelona and is displayed in Fig. 3 (b).

The third system is a Sun Fire X4600 M2. It comprises 8 AMD Opteron 8218 (Santa Rosa) dual-core processors with 1 MB L2 cache per core. Each CPU has access to 8 GB local memory and three HyperTransport interfaces. We will refer to this system as X4600. The precise system architecture can be found in [12].

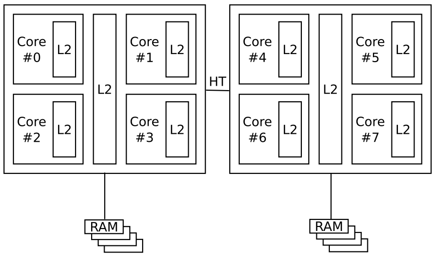
The fourth system is made of four Intel Xeon X7460 and will be referred to as Dunnington. The Dunnington is a hexa-core processor with a 16 MB L3 cache shared by all cores. Furthermore, the cores are grouped into pairs with an L2 cache of 3 MB per pair. The system is equipped with 32 GB of main memory.

The last system is an Intel Nehalem pre-production system. It consists of two Intel Xeon X5570 running at 2,9 GHz and 12 GB DDR3 memory. The CPU has

[1] http://userweb.kernel.org/~kzak/util-linux-ng



(a) Intel Clovertown system



(b) AMD Barcelona system

Fig. 3. System architectures of test machines

four cores (HyperThreading disabled), an 8 MB L3 cache and two QPI interfaces. This NUMA machine will be referred to as Nehalem.

Matrix I was used on Clovertown, Barcelona and Nehalem, whereas matrix II was used on X4600 and Dunnington. This applies to all results presented below. Also we do not report timings for the vector scaling operations in MLEM as these operations need less time than inter-process communication.

5.1 Striping

Subdiving matrix blocks into vertical stripes proved beneficial on every system. Most systems showed best performance with stripes of 200 000 columns in width. This corresponds to sections of approximately 0.8 MB in the input vector (forward projection) and output vector (back projection) of the SpMV operation.

The X4600 with its small L2 cache of 1 MB showed exceptional good speedup with even smaller stripes. For example, the time per iteration step on a single core dropped form 185 s to 52 s when using 16 vertical stripes. This corresponds to a vector section length of 200 KB, which easily fits into the L2 cache.

5.2 Load Balancing

The bisection approach generally gives good load balancing, as can be seen in Fig. 4. The chart displays the relative load imbalance, which we define as

$$L_{\mathrm{rel}} = \frac{\max_i |t_i - t_{\mathrm{avg}}|}{t_{\mathrm{avg}}} \quad .$$

The load imbalance was measured in runs with eight processes and averaged over five iteration steps. Fig. 4 also shows that striping improves load balancing, especially on the X4600 system. We attribute the remaining load imbalance to the fact that partitioning is based on the number of matrix elements per block. A more sound fundament of block size would be the number of cache misses in the source and destination vectors. But this approach would require dynamic load balancing by repartitioning the large matrix after the first iteration steps, which is inefficient.

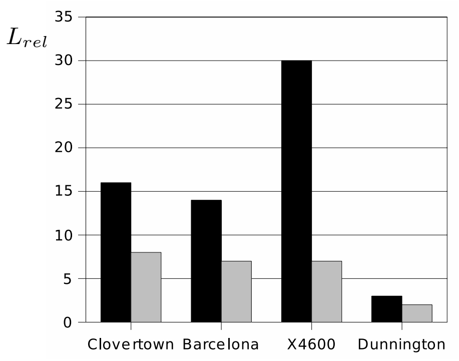


Fig. 4. Relative load imbalance with striping switched off (black) and on (gray)

5.3 Core Pinning

The selection of CPU cores is essential for good speedup. The core numbers in Fig. 5 correspond to those in Fig. 3. FP and BP denote forward and back projection, respectively. Total time also encompasses the time needed for communicating the partial results. Taking the Clovertown system as an example it can be seen, that pinning two processes to cores 0 and 2, which share a common L2 cache, returns a speedup of only 1.1, whereas using cores 0 and 1, results in a speedup of 1.8. On NUMA machines it is advantageous to use cores with separate memory links (see Fig. 5 (b)). In test runs with two processes this improved the speedup by about 25% on the Barcelona system.

N	Cores	FP [s]	BP [s]	Total [s]	Speed-up
1	0	3.75	3.89	7.64	
2	0,2	3.12	3.58	6.76	1.1
2	0,1	2.11	2.13	4.30	1.8
4	0,2,4,6	1.28	1.31	2.99	2.6
4	0,1,4,5	0.83	0.85	1.95	3.9

(a) Clovertown

N	Cores	FP [s]	BP [s]	Total [s]	Speed-up
1	0	6.43	5.79	12.23	
2	0,4	3.81	3.77	7.65	1.6
2	0,1	3.23	2.90	6.17	2.0
4	0,1,4,5	1.78	1.76	4.07	3.0
4	0,1,2,3	1.50	1.33	3.16	3.9

(b) Barcelona

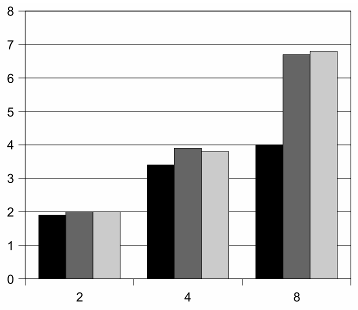
Fig. 5. Different alternatives for process to core pinning

5.4 Speedup

Finally, we present speedup data for all five test machines. We used the best pinning and optimal striping for these runs. Fig. 6 compares the speedup on Clovertown, Barcelona and Nehalem, using the smaller matrix I. The column labeled "Comm." gives the communication time for forward (FP) and back projection (BP), respectively. It can be seen, that Clovertown's system architecture only scales well up to two processes. With more processes running, the memory bandwidth of the front-side bus is saturated. The final speedup of 4.0

N	FP [s]	Comm. [s]	BP [s]	Comm. [s]	Total [s]	Speed-up
1	2.68	0.0	2.61	0.0	5.29	
2	1.37	0.01	1.35	0.02	2.75	1.9
4	0.73	0.05	0.71	0.04	1.54	3.4
8	0.57	0.11	0.58	0.08	1.34	4.0

(a) Timings on Clovertown

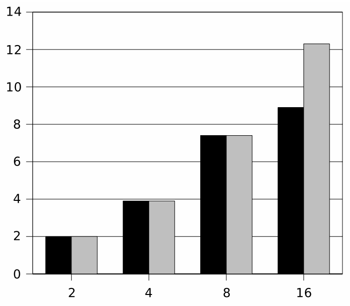


(b) Speedup on Clovertown (black), Barcelona (darkgray) and Nehalem (lightgray)

Fig. 6. Results for Clovertown, Nehalem and Barcelona, using matrix I

N	FP [s]	Comm. [s]	BP [s]	Comm. [s]	Total [s]	Speed-up
1	26.67	0.0	25.66	0.0	52.33	
2	13.28	0.13	12.81	0.12	26.34	2.0
4	6.66	0.22	6.43	0.23	13.54	3.9
8	3.38	0.22	3.29	0.21	7.10	7.4
16	2.34	0.63	2.33	0.61	5.91	8.9

(a) Timings on X4600



(b) Speedup on X4600 (black) and Dunnington (gray)

Fig. 7. Results for X4600 and Dunnington, using matrix II

when using eight cores can easily be outperformed by Barcelona with a speedup of 6.8.

The X4600 system scales reasonably well up to 8 cores, i.e. as long as we only use one core per chip (Fig. 7 (a)). The final speedup is 8.9, whereas Dunnington reaches a speedup of 12.3 on 16 cores.

6 Conclusion and Outlook

In this paper we applied two partitioning techniques, recursive bisection and striping, to the SpMV operations in MLEM. The techniques provided good load balancing and cache behavior. We also showed the importance of pinning MPI processes to cores on multicore architectures. There is ongoing work to automate this process [9]. Our approach reduces memory requirements of large matrices by using data parallelism and distributing matrix blocks to processor cores. A considerable amount of memory is also saved by our new matrix format. Future work will focus on improving cache usage, by reordering rows and columns of the system matrix.

References

1. Chen, C.M., Lee, S.-Y., Cho, Z.H.: Parallelization of the EM algorithm for 3-D PET Image Reconstruction. IEEE Transactions on Medical Imaging 10(4), 513–522 (1991)
2. Desjardins, B., Lecomte, R.: Parallel Approach to Iterative Tomographic Reconstruction for High Resolution PET Imaging. In: IEEE Nuclear Science Symposium, vol. 2, pp. 1551–1555 (1997)
3. Goumas, G., Kourtis, K., Anastopoulos, N., Karakasis, V., Koziris, N.: Understanding the Performance of Sparse Matrix-Vector Multiplication. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 283–292 (2008)
4. Minde, J., Weidendorfer, J., Klug, T., Trinitis, C.: PET-Bildrekonstruktion auf der Cell-BE. In: Tagungsband Kommunikation in Clusterrrechnern und Clusterverbundsystemen, Tagung, RWTH Aachen, Aachen, Germany, vol. 3 (2007)
5. Ni, J., Li, X., Wang, G.: Review of Parallel Computing Techniques for Computed Tomography Image Reconstruction. Current Medical Imaging Reviews 2(4), 405–414 (2006)
6. Picard, Y., Selivanov, V., Verreault, M., Lecomte, R.: Optimizing Communications for Parallel ML-EM Image Reconstruction on Large Clusters of Processors. In: Conference Record of the IEEE Nuclear Science Symposium, vol. 3, pp. 1574–1580 (1998)
7. Qi, J., Leahy, R.M.: Iterative reconstruction techniques in emission computed tomography. Physics in Medicine and Biology 51, 541–578 (2006)
8. McElroy, D.P., Hoose, M., Pimpl, W., Spanoudaki, V., Schüler, T., Ziegler, S.I.: A true singles list-mode data acquisition system for a small animal PET scanner with independent crystal readout. Physics in Medicine and Biology 50, 3323–3335 (2005)
9. Ott, M., Klug, T., Weidendorfer, J., Trinitis, C.: Autopin - Automated Optimization of Thread-to-Core Pinning on Multicore Systems. In: First Workshop on Programmability Issues for Multi-Core Computers, Gothenburg, Sweden (2008)
10. Schirmer, J., Torres-Espallardo, I., Minde, J., Spanoudaki, V., Trager, R., Ziegler, S.: Analytic and Monte Carlo Derived System Matrices for Small Animal PET Imaging. In: Annual Congress of the European Association of Nuclear Medicine, Munich, Germany (2008)
11. Shepp, L.A., Vardi, Y.: Maximum likelihood reconstruction for emission tomography. IEEE Transactions on Medical Imaging MI-1(2), 113–122 (1982)
12. Sun Microsystems: Sun Fire X4600 M2 Server Architecture, White Paper, http://www.sun.com/servers/x64/x4600/arch-wp.pdf

Experience with Approximations in the Trust-Region Parallel Direct Search Algorithm*

S.M. Shontz[1,**], V.E. Howle[2], and P.D. Hough[3]

[1] Department of Computer Science and Engineering
The Pennsylvania State University,
University Park, PA 16802
shontz@cse.psu.edu

[2] Department of Mathematics and Statistics
Texas Tech University Lubbock, TX 79409
victoria.howle@ttu.edu

[3] Advanced Software Research and Development Department
Sandia National Laboratories
Livermore, CA 94551
pdhough@sandia.gov

Abstract. Recent years have seen growth in the number of algorithms designed to solve challenging simulation-based nonlinear optimization problems. One such algorithm is the Trust-Region Parallel Direct Search (TRPDS) method developed by Hough and Meza. In this paper, we take advantage of the theoretical properties of TRPDS to make use of approximation models in order to reduce the computational cost of simulation-based optimization. We describe the extension, which we call mTRPDS, and present the results of a case study for two earth penetrator design problems. In the case study, we conduct computational experiments with an array of approximations within the mTRPDS algorithm and compare the numerical results to the original TRPDS algorithm and a trust-region method implemented using the speculative gradient approach described by Byrd, Schnabel, and Shultz. The results suggest new ways to improve the algorithm.

Keywords: approximation models, parallel optimization, nonlinear programming.

1 Introduction

Coupling optimization software with simulations has become a common way to address optimal design and analysis questions. For example, one may want to

* Most of this work was performed while all of the authors were members of the Computational Sciences and Mathematics Research group at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

** Part of this work was performed while the first author was a member of the Center for Applied Mathematics at Cornell University, supported by the National Physical Science Consortium.

identify parameters for a new model such that simulation results most closely match experimental results or to determine an optimal device design. It is well-known that simulation-based optimization problems pose many challenges. We consider gradient-based methods in which finite-difference approximations to the gradient are used because analytic gradients are not available. We consider problems with a small number of variables, so the dominant optimization cost is the function evaluation cost. Our goal is to reduce the number of function evaluations performed by leveraging parallelism and approximation models that incur less computational expense.

There are many ways of parallelizing optimization algorithms such as [1]-[5]. We focus on two variants of trust-region optimization methods. The first is a speculative gradient technique introduced in [6]. The key assumption of this method is that in a classical Newton method (with either line search or trust region), the initial trial point at each iteration is usually accepted. Since small clusters of processors are commonly available, the additional processors can be used to begin computing finite-difference gradient components at the trial point while the function is being evaluated. If the trial point is rejected, nothing is lost. However, since it is usually accepted, this approach results in substantial computational savings. The other approach we employ is the Trust-Region Parallel Direct Search (TRPDS) algorithm developed in [7]. This method combines the trust-region version of a Newton method with a parallel direct search (PDS) method in a way that retains the best properties of both. In particular, PDS is used to augment the set of search directions to offset inaccuracies in the numerical gradient approximations. Since PDS is inherently parallel, this can be done without any additional cost when multiple processors are available.

Approximation models are another approach to reducing the overall cost of the optimization. There are many ways in which an approximation to a high-accuracy model can be constructed, such as response surface and spacing mapping techniques for constructing surrogates in [9]. Examples of optimization strategies that make use of approximations are described in [10]-[13]. In this work, we consider the use of approximation models within the TRPDS algorithm. Unlike [10] and [13], this approach incorporates the use of numerical gradients. In addition, we do not assume that trial iterates satisfy decrease conditions by construction as is true in classical trust-region methods and in [12]. Our work is related to [11] in that both fall into a general class of trust-region algorithms described in [8], and both leverage the flexibility of this class of algorithms by using approximation models to reduce computational cost. The primary distinction is that our approach seeks to find decrease quickly at each iteration using the approximation rather than optimizing the approximation at each iteration as in [11]. We will present a case study using TRPDS in conjunction with approximations obtained by reducing accuracy of the physics model and by constructing quadratic representations. Results suggest ways of improving the algorithm.

2 Speculative Gradients

Recall that a trust-region method is an optimization method in which each iteration entails constructing a quadratic Taylor series expansion of the function and minimizing that expansion over a region in which it is expected to be a good approximation to the function. We refer the reader to standard references like [19] for details, but we present a summary of the speculative gradient version of the algorithm here.

We first note that a function gradient is required to construct each Taylor series expansion and is computed using finite-difference calculations since analytic gradient information is not available. Furthermore, let us assume that we have p processors at our disposal. For simplicity, the algorithm is described assuming that a function evaluation uses one processor and forward/backward finite-differences are being performed. Generalizing to multiprocessor function evaluations and central differences is straightforward. To maximize the use of available processors, one processor is used to evaluate the trial point, and the remaining $p-1$ processors are used to calculate up to $p-1$ components of the finite-difference gradient. If the trial point is accepted, we have $p-1$ components of the gradient available and only need to calculate the remaining $n-(p-1)$ components, where n is the problem dimension. If the trial point is not accepted, no time is lost because the function evaluation is required regardless. The speculative-gradient trust-region algorithm is shown below in Algorithm 1. Here $\mathbf{x}_k$ is the current iterate, $\mathbf{s}$ is the trial step, $g(\mathbf{x}_k)$ is the gradient of f at the current point, $H_k \approx \nabla^2 f(\mathbf{x}_k)$ is the Hessian approximation at the current point, δ_k is the size of the trust region, and

$$\psi(\mathbf{s}) = g(\mathbf{x}_k)^T \mathbf{s} + \frac{1}{2}\mathbf{s}^T H_k \mathbf{s}. \tag{1}$$

Algorithm 1. Trust-Region Method with Speculative Gradients

```
Given $p$ processors, $\mathbf{x}_0$, $\mathbf{g}_0$, $H_0$, $\delta_0$, and $\eta \in (0,1)$
**for** $k = 0, 1, \ldots$ until convergence **do**
    **for** $i = 0, 1, \ldots$ until step accepted **do**
        1. Find $\mathbf{s}_i$ that approximately solves the quadratic subproblem
        2. Processor 0: evaluate $f(\mathbf{x}_k + \mathbf{s}_i)$
           Processor $j$: evaluate $g_{j-1}(\mathbf{x}_k + \mathbf{s}_i)$ for $j = 1, \ldots, p-1$
        3. Compute $\rho = (f(\mathbf{x}_k + \mathbf{s}_i) - f(\mathbf{x}_k))/\psi(\mathbf{s}_i)$
        **if** $\rho > \eta$ **then**
            4. Accept step, set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_i$
            5. Processor $j$: for $j = 0, \ldots, p-1$:
            **for** $l = j+1, \ldots, n-(p-1)$, step $p$, **do**
                6. Evaluate $g_{l+(p-1)}(\mathbf{x}_k + \mathbf{s}_i)$
            7. Update $H_k$
        **else**
            8. Reject step
        9. Update $\delta_k$
```

3 TRPDS with Generalized Approximation Models

In 2002, Hough and Meza developed the Trust Region-Parallel Direct Search (TRPDS) algorithm [7]. TRPDS employs the trust-region framework but uses the PDS algorithm of Dennis and Torczon [1] to solve a non-standard subproblem, the PDS subproblem, at each iteration. Solving this subproblem entails using PDS to minimize the function itself subject to the trust-region constraint and a fraction of Cauchy decrease constraint. See [7] for more details.

The original motivation for TRPDS was to combine the desirable convergence properties of trust-region methods with the robustness of PDS for low-accuracy functions. However, this algorithmic combination has a great deal of additional flexibility, which we leverage by extending the subproblem solution to a two-phase approach that incorporates the use of an approximation model. The first phase consists of using PDS to find the j best solutions to the following problem:

$$\begin{aligned} \min_{\mathbf{s}\in\mathbb{R}^n} \quad & m(\mathbf{x}_k+\mathbf{s}) \\ \text{s. t.} \quad & \|\mathbf{s}\|_2 \le 2\delta_k, \\ & \psi(\mathbf{s}) \le \beta\|g(\mathbf{x}_k)\| \min\left(\delta_k, \frac{\|g(\mathbf{x}_k)\|}{C}\right), \end{aligned} \tag{2}$$

where j is an integer, m is a computationally inexpensive approximation to the objective function, $\mathbf{x}_k$ is the current iterate, $\mathbf{s}$ is the trial step, δ_k is the size of the trust region, $\beta > 0$, $C > 0$, and $\psi(\mathbf{s})$ is defined in (1). This resembles the PDS subproblem except the objective function has been replaced by an approximation model. Also, the constraint on $\psi(\mathbf{s})$ enforces the fraction of Cauchy decrease condition. We are using p processors and are taking $j = p$ for simplicity of description. In the second phase, each processor evaluates the objective function at one of these j trial points. The point that yields the lowest function value is returned to and processed by the trust-region framework. This variation of TRPDS, referred to as mTRPDS, is given in Algorithm 2 below.

```
Algorithm 2. $m$TRPDS
Given $p$ processors, $\mathbf{x}_0$, $\mathbf{g}_0$, $H_0$, $\delta_0$, and $\eta \in (0,1)$
for $k = 0, 1, \ldots$ until convergence do
    1. Solve $H_k\mathbf{s}_N = -\mathbf{g}_k$
    for $i = 0, 1, \ldots$ until step accepted do
        2. Form an initial simplex using $\mathbf{s}_N$
        3. Compute the $p$ best approximate solutions $\mathbf{s}_1,\ldots,\mathbf{s}_p$ to (2) using PDS
        4. Determine $\mathbf{s} \in \{\mathbf{s}_1, \ldots, \mathbf{s}_p\}$ that minimizes $f(\mathbf{x}_k + \mathbf{s})$
        5. Compute $\rho = (f(\mathbf{x}_k + \mathbf{s}_i) - f(\mathbf{x}_k))/\psi(\mathbf{s}_i)$
        if $\rho > \eta$ then
            6. Accept step and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_i$, evaluate $\mathbf{g}_{k+1}$, $\mathbf{H}_{k+1}$
        else
            7. Reject step
        8. Update $\delta$
```

In [7], it was observed that the TRPDS class of algorithms fits into the generalized trust-region framework of Alexandrov et al. [8], which provides much flexibility in the choice of trust-region model and in the step computation. In particular, let a be an approximation to the objective function, f. If $a(\mathbf{x}_k) = f(\mathbf{x}_k)$, and $\nabla a(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)$, and the sequence of iterates generated during the optimization satisfies a fraction of Cauchy decrease condition according to a, then the standard trust-region convergence theory implies this class of methods will converge to a local minimizer of f [8]. Note that a is different from the approximation, m, described in mTRPDS. The former could be any model that satisfies the conditions above; however in mTRPDS, as in TRPDS, we fix $a = \psi$, where ψ is defined in (1). Furthermore, the fraction of Cauchy decrease condition is enforced by the second constraint in (2), so the iterates generated by mTRPDS also satisfy the above assumptions. Thus, as with TRPDS, mTRPDS is guaranteed to converge according to the theory in [8].

Now that we have established the convergence properties of our TRPDS modification, we present a case study for an earth penetrator design problem.

4 Case Study for Earth Penetrator Design

To evaluate the mTRPDS algorithm, we consider two problems in earth penetrator design, a problem of long-standing interest to the Departments of Energy and Defense. We consider the scenario in which there is a pre-existing hole in the target, as depicted in Fig. 1.

The penetrator model is fairly simple in that we consider it to be a solid "egg" made of one material. The penetrator shaft is divided into three sections, and their lengths are varied independently. The radius is held constant. In the first problem, we wish to find lengths that minimize the maximum acceleration

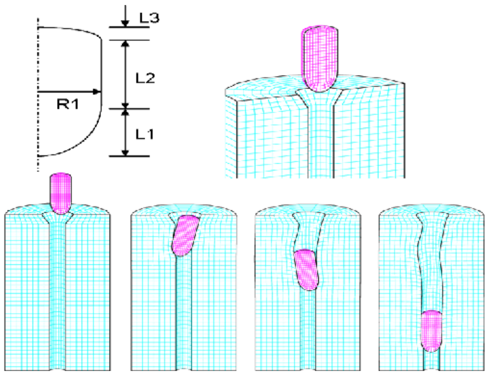


Fig. 1. The earth penetrator radius is held fixed, while the lengths are varied independently. The goal is to find section lengths that will optimize mission performance.

subject to bounds on the length parameters. Our optimization problem is the following:

$$\begin{aligned} \min_{\mathbf{L}\in \mathbb{R}^3} \; & F(\mathbf{L}) = \max(\text{acceleration}) \\ \text{s.t.} \; & l_i \le L_i \le u_i, i = 1\ldots 3, \end{aligned} \tag{3}$$

where $\mathbf{L}$ is the vector containing the three unknown length parameters, L_i, and l_i and u_i are the lower and upper bounds, respectively. In the second problem, we wish to find lengths resulting in maximal penetration depth subject to bounds on the length parameters. Our optimization problem is the following:

$$\begin{aligned} \min_{\mathbf{L}\in \mathbb{R}^3} \; & F(\mathbf{L}) = -(\text{depth of penetration}) \\ \text{s.t.} \; & l_i \le L_i \le u_i, i = 1\ldots 3. \end{aligned} \tag{4}$$

Presto, a Sandia-developed three-dimensional explicit transient dynamics code that is implemented using Lagrangian finite elements [14], is used to model the mechanical deformation of the penetrator upon impact. The ACME library [15] is used for the contact algorithms. We use a finite element model that represents a solid, homogeneous body. The penetrator is modeled as an elastic material, and a Mohr-Coulomb soil constitutive model is used to represent the target. The penetrator and target models are axisymmetric. CUBIT [16] was used to develop a parametric mesh model that was used to generate the finite element mesh for each set of length parameters. Meshes consist of eight-node hex elements, and the time step is chosen to satisfy the Courant stability condition.

To compare the approaches, we ran a set of computational experiments on a Linux cluster with dual 3.6 GHz Intel EM64T processors with 6 GB RAM. Each node runs Red Hat Enterprise Linux WS 4 and MPICH over an Infiniband network. For mTRPDS, we chose a range of approximations constructed in the following ways: 1) altering the mesh discretization, 2) altering the amount of event time simulated, and 3) using a Taylor series to construct a quadratic model of the function. The results were compared to those obtained using TRPDS and the trust-region method with speculative gradients. In all cases, the gradient was approximated by central differences, and a BFGS approximation to the Hessian was employed.

We chose the number of processors to be that which is ideal for speculative gradient computation. The penetrator design problem has three variables, so seven function evaluations need to be done simultaneously to compute the objective and central difference gradient at the trial iterate. We used 16 processors for each simulation and one for the optimization process, totaling 113 processors. Thus, we also used 113 processors for the mTRPDS computational experiments. The ideal settings for mTRPDS on 113 processors are a search pattern size (sps) of 7 and $j = 7$. The optimization algorithms are implemented in OPT++, and additional algorithmic parameters were set to their default values shown in [17].

Table 1 shows the set of algorithms and models used in the experiments and the wall clock time for a single execution of each model. The approximation models were named according to the following convention: MeshXk refers to the

Table 1. This table shows the algorithm-model combinations used in experiments and the time required for a single execution of each model. SpecGrad and TRPDS use the truth model, where mTRPDS uses the truth model along with the specified approximation models.

Key	Algorithm	Model	Time for Single Model Execution
1	SpecGrad	Truth (Mesh640k, Time25ms)	$2 - 3$ hours
2	TRPDS	Truth (Mesh640k, Time25ms)	$2 - 3$ hours
3	mTRPDS	QuadraticModel	negligible
4	mTRPDS	Mesh10k	$0.8 - 1.3$ hours
5	mTRPDS	Mesh80k	$1.3 - 1.8$ hours
6	mTRPDS	Time6.25ms	$1.1 - 1.6$ hours
7	mTRPDS	Time12.5ms	$1.7 - 2.3$ hours

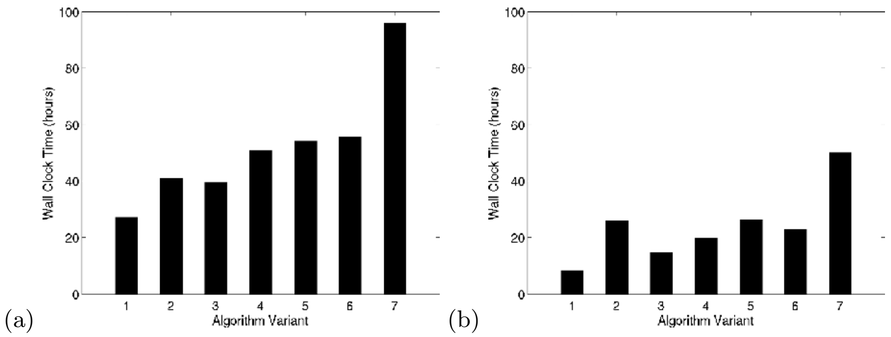


Fig. 2. Key is given in Table 1. (a) This figure shows the wall clock time required to achieve a 0.1% change in the function value for (3). Variations in results are due primarily to different numbers of iterations. (b) This figure shows the wall clock time required to achieve an 0.1% change in the function value for (4). Variations in results are due primarily to different costs per iteration.

model using a mesh with X thousand elements; TimeYms refers to the model in which the event time simulated is Y milliseconds, and QuadraticModel is the quadratic Taylor series expansion for f. The results of the experiments appear in Figs. 2 through 4. Figure 2 shows the wall clock times to reach a 0.1% change in the function value for the algorithms and approximations tested. We use this criteria because investigation into behavior of the algorithms after this point uncovered the need for further research into useful numerical stopping criteria.

For problem (4), shown in Fig. 2b, all experiments took approximately the same number of iterations. The variations in wall clock times, therefore, are due primarily to differences in the average wall clock time per iteration. We see from Fig. 3b that there can be a notable difference in time per iteration. Further investigation into these differences revealed opportunities to improve the computational efficiency of the mTRPDS algorithm. For the depth of penetration, the

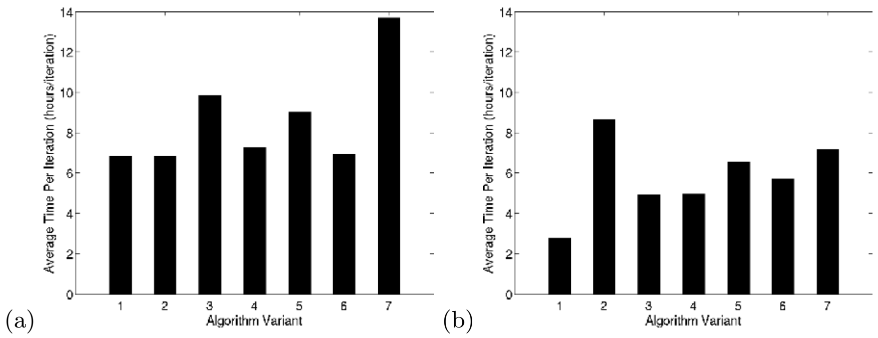


Fig. 3. Key is given in Table [1]. (a) This figure shows the average time per iteration for ([3]). Results suggested a need for better characterization of algorithm and approximation performance. (b) This figure shows the average time per iteration for ([4]). Results suggested specific improvements in computational efficiency for mTRPDS.

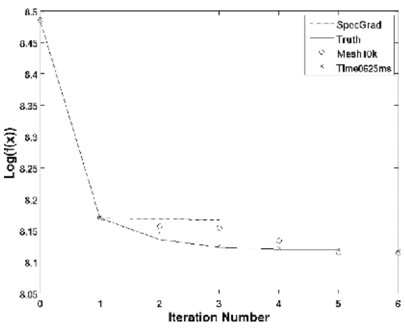


Fig. 4. Log of function value vs. iteration for the four algorithms with comparable average time per iteration for problem ([3]). TRPDS-based algorithms move to a solution with a lower function value, thereby taking more iterations.

approximation models track the truth fairly well (see [18]). This indicates that we should reduce the value of j and incorporate the computation of speculative finite-difference gradients for those j points. It is possible that the best choice is $j = 0$. This would reduce the number of truth evaluations needed at each iteration, thereby reducing the total time. We also found that PDS was doing approximation evaluations that make little or no apparent contribution to the progress of the optimization algorithm. Even though PDS is operating on the approximation models, this can add up to notable time. We would like to eliminate that by developing a dynamic scheme for managing the amount of work done by PDS based on the quality of the approximation evaluations it does.

Further investigation into the results for problem ([3]) show that the primary difference in wall clock times (shown in Fig. [2]a) is due to the algorithms taking different numbers of iterations. We see in Fig. [3]a, however, that several algorithm-model combinations have comparable average times per iteration. To get further insight, we plot the function value versus iteration number for those variants in Fig. 4. We see that the TRPDS-based algorithms start out the same as the speculative gradient algorithm but then move toward solutions with lower function values, thereby taking longer. To better understand the reasons for this behavior, further characterization of the effects of problem features on algorithm performance is needed. Such characterization requires a set of computationally expensive, physics-based test problems. Test problems with these characteristics are hard to come by, as standard test sets do not meet these criteria.

5 Conclusions and Future Work

We have extended the TRPDS algorithm of Hough and Meza to include the use of an approximation model in solving the PDS subproblem. This approach, which we call mTRPDS, uses the approximation to identify several candidates for the trial iterate and takes advantage of parallel processing to evaluate them. The algorithm was studied, together with TRPDS and a speculative gradient implementation of the classical trust-region method, in the context of two earth penetrator optimal design problems but is generally applicable to any simulation-based unconstrained or bound-constrained nonlinear optimization problem.

The empirical results we collected from the numerical tests suggested both short-term, concrete areas for improving the computational efficiency of mTRPDS and longer-term research areas. To improve computational efficiency, we will further examine the choice of j to reduce the number of truth evaluations needed. We will also develop a means of dynamically managing the amount of work PDS performs using the approximation model. Longer-term research includes developing meaningful numerical stopping criteria for optimization algorithms and characterizing the effects of problem characteristics on algorithm performance.

Acknowledgments. The authors wish to thank John Dennis, Juan Meza, and Virginia Torczon for many helpful discussions and suggestions on the TRPDS-based algorithms. They would also like to thank Monica Martinez-Canales, Genetha Gray, Laura Swiler, Mike Chiesa, and Randy Settgast for assistance in implementing and understanding the earth penetrator application.

References

1. Dennis Jr., J.E., Torczon, V.: Direct Search Methods on Parallel Machines. SIAM J. Optimiz. 1(4), 448–474 (1991)
2. Ingber, L.: Simulated Annealing: Practice Versus Theory. Math. Comput. Model. 18(11), 29–57 (1993)
3. Laarhoven, P.J.M.: Parallel Variable Metric Algorithms for Unconstrained Optimization. Math. Program. 33, 68–81 (1985)
4. Phua, P.K.-H., Zeng, Y.: Parallel Quasi-Newton Algorithms for Large-Scale Optimization. Tech. Report TRB2/95, National Univ. of Singapore, Singapore (1995)

5. Straeter, T.A.: A Parallel Variable Metric Optimization Algorithm. Tech. Report NASA TN D-7329, NASA, Langley Research Center, Hampton, VA (1973)
6. Byrd, R.H., Schnabel, R.B., Shultz, G.A.: Parallel Quasi-Newton Methods For Unconstrained Optimization. Math. Program. 42, 273–306 (1988)
7. Hough, P.D., Meza, J.C.: A Class of Trust-Region Methods for Parallel Optimization. SIAM J. Optimiz. 13(1), 264–282 (2002)
8. Alexandrov, N., Dennis Jr., J.E., Lewis, R.M., Torczon, V.: A Trust Region Framework for Managing the Use of Approximation Models in Optimization. J. Struct. Optimiz. 15(1), 16–23 (1998)
9. Dennis Jr., J.E.: Surrogate Modelling and Space Mapping for Engineering Optimization: A Summary of the Danish Technical University November 2000 Workshop. Tech. Report CAAM-TR00-35, Rice University, Houston, TX (2000)
10. Booker, A.J., Dennis Jr., J.E., Frank, P.D., Serafini, D.B., Torczon, V., Trosset, M.W.: A Rigorous Framework for Optimization of Expensive Functions by Surrogates. J. Struct. Optimiz. 17(1), 1–13 (1999)
11. Giunta, A.A., Eldred, M.S.: Implementation of a Trust Region Model Management Strategy in the DAKOTA Optimization Toolkit. In: Proceedings of 8th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, CA (2000)
12. Nash, S.G.: A Multigrid Approach to Discretized Optimization Problems. Optim. Method. Softw. 14, 99–116 (2000)
13. Siefert, C.M.: Model-Assisted Pattern Search. Honors Thesis, Department of Computer Science, College of William & Mary, Williamsburg, VA (2000), http://www.cs.wm.edu/~va/CS495/siefert.ps.gz
14. Koteras, J.R., Guillerud, A.S., Crane, N.K., Hales, J.D., Reinert, R.K.: Presto Users Guide 2.7. Tech. Report SAND2007-3749, Sandia National Laboratories, Albuquerque, NM (2007)
15. Brown, K.H., Summers, R.M., Glass, M.W., Guillerud, A.S., Heinstein, M.W., Jones, R.E.: ACME: Algorithms for Contact in a Multiphysics Environment, API Version 1.0. Tech. Report SAND2001-3318, Sandia National Laboratories, Albuquerque, NM (2001)
16. Owen, S.J.: CUBIT Geometry and Mesh Generation Toolkit (2006), http://www.cubit.sandia.gov
17. Meza, J.C., Hough, P.D., Williams, P.J., Oliva, R.A.: OPT++ 2.4 Documentation (2007), http://csmr.ca.sandia.gov/opt++/opt++2.4_doc/html/index.html
18. Martinez-Canales, M.L., Swiler, L.P., Hough, P.D., Gray, G.A., Chiesa, M.L., Heaphy, R., Thomas, S.W., Trucano, T.G., Lee, H.K.H., Taddy, M., Gramacy, R.B.: Penetrator Reliability Investigation and Design Exploration. Tech. Report SAND2006-7669, Sandia National Laboratories, Livermore, CA (2006)
19. Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. Academic Press, London (1981)

A 3D Vector-Additive Iterative Solver for the Anisotropic Inhomogeneous Poisson Equation in the Forward EEG problem

Vasily Volkov[1], Aleksei Zherdetsky[1], Sergei Turovets[2], and Allen Malony[2]

[1] Department of Mathematics and Mechanics, Belarusian State University, 4 Independence Ave., Minsk 220050, Republic of Belarus
volkovvm@bsu.by
[2] NeuroInformatics Center, 5294 University of Oregon, Eugene, OR 97403, USA
(sergei,malony)@cs.uoregon.edu

Abstract. We describe a novel 3D finite difference method for solving the anisotropic inhomogeneous Poisson equation based on a multi-component additive implicit method with a 13-point stencil. The serial performance is found to be comparable to the most efficient solvers from the family of preconditioned conjugate gradient (PCG) algorithms. The proposed multi-component additive algorithm is unconditionally stable in 3D and amenable for transparent domain decomposition parallelization up to one eighth of the total grid points in the initial computational domain. Some validation and numerical examples are given.

1 Introduction

The challenge in most tomographic techniques is to determine unknown complex coefficients or driving sources in the partial differential equations (PDEs) governing the physics of the particular experimental modality. Problems in neuroscience such as electroencephalography (EEG) and magnetoencephalograpy (MEG) source localization, electrical impedance tomography (EIT) or diffuse optical tomography (DOT) are inherently non-linear, underdetermined and ill-posed, requiring high accuracy in measurements and PDE inverse modeling [1]. The first step in solving such inverse problems is to find a numerical method to solve the direct (forward) problem. When the physical model is three-dimensional and geometrically complex, like the human brain, the high- resolution forward solution can be difficult to construct and compute.

Until recently, most practical research in this field has opted for simplistic analytical or semi-analytical models of a human head in the forward calculations [2]. With geometric information becoming more readily available from MRI or CT scans, finite element (FE) and finite difference (FD) approaches can now incorporate realistic 3D head geometry for human head model construction. However, most of the published models, with a few exceptions, treat the human head tissues as isotropic, while it is well known that brain white matter, skull and facial/scalp muscles are highly anisotropic, with the anisotropic ratio estimated to be between 1:3 and 1:10 [3 and references therein].

In the present study we propose an algorithm for solving the anisotropic diffusion equation based on multi-component (vector-additive) implicit FD methods. Not only are these methods unconditionally stable in 3D, but they offer the potential for high domain decomposition parallelization, and are promising candidates for computational acceleration with GPGPUs (general purpose graphics processing units) [4]. We introduce the algorithm and assess the serial performance of the proposed method in comparison with the most efficient solvers from the family of the preconditioned conjugate gradient (PCG) algorithms.

2 Statement of the Problem

The relevant frequency spectrum in EEG, MEG and EIT of the human head is typically below 1 kHz, and most studies deal with frequencies between 0.1 and 100 Hz. Therefore, the physics of EEG/MEG can be well described by the quasi-static approximation of the Maxwell equations, the Poisson equation. The electrical forward problem can be stated as follows: given the positions, orientations and magnitudes of dipole current sources, $\varphi(x, y, z)$, as well as geometry and electrical conductivity of the head volume (Ω), calculate the distribution of the electrical potential on the surface of the head (scalp) (Γ_Ω). Mathematically, it means solving the inhomogeneous anisotropic Poisson equation [2]:

$$\nabla \bullet (\sigma (\nabla u) = \varphi(x, y, z), \text{ in } \Omega \tag{1}$$

with no-flux Neumann boundary conditions on the scalp:

$$\sigma (\nabla u) \bullet n = 0, \text{ on } \Gamma_\Omega. \tag{2}$$

Here $\sigma = \sigma_{ij}(x,y,z)$ is an inhomogeneous symmetric tensor of the head tissues conductivity. Having computed potentials $u(x,y,z)$ and current densities $J=- \sigma (\nabla u)$, the magnetic field B can be found through the Biot-Savart law. The similar non-stationary anisotropic diffusion equation is relevant also in the DOT forward problem modeling [1] and the white matter tractography studies using diffusion tensor MRI imaging [5].

Previously, we built an iterative finite difference forward problem solver for an isotropic version of (1) and (2) based on the multi-component alternating directions implicit (ADI) algorithm [6]. It is a generalization of the classic ADI algorithm, but with improved stability in 3D (the multi-component FD ADI scheme is unconditionally stable in 3D for any value of the time step [7,8]). To describe the electrical conductivity in the heterogeneous biological media within arbitrary geometry, the method of the embedded boundaries has been used. Here an object of interest is embedded into a cubic computational domain with extremely low conductivity values in the external complimentary regions modeling the surrounding air. This effectively guarantees there are no current flows out of the physical area (the Neumann boundary conditions, (2), is naturally satisfied). The idea of the iterative implicit method is to find the solution of (1) and (2) as a steady state of the appropriate evolution (diffusion) problem. At every iteration step, the spatial operator is split into the sum of three 1D operators, which are evaluated alternatively at each sub-step. Such a scheme is accurate to $O[\tau+(\Delta x)^2)+(\Delta y)^2+(\Delta z)^2]$. In contrast with the classic ADI method, the multi-component ADI uses the regularization (averaging) for evaluation of the variable at the previous instant of time.

Parallelization of the vector-additive ADI algorithm in a shared memory multiprocessor environment (OpenMP) is straightforward, as it consists of nests of independent loops over “bars” of voxels for solving the effective 1D problem in every iteration. However, it is less suitable for implementation in an environment with a distributed memory. In the next section we present a vector-additive algorithm of the domain decomposition type which is potentially amenable for implementation at greater parallel degree.

3 Numerical Scheme

In the Cartesian coordinate system, (1) is expressed as

$$\frac{\partial}{\partial x}\left(\sigma_{xx}\frac{\partial u}{\partial x}+\sigma_{xy}\frac{\partial u}{\partial y}+\sigma_{xz}\frac{\partial u}{\partial z}\right)+\frac{\partial}{\partial y}\left(\sigma_{yy}\frac{\partial u}{\partial y}+\sigma_{yx}\frac{\partial u}{\partial x}+\sigma_{yz}\frac{\partial u}{\partial z}\right)+$$
$$+\frac{\partial}{\partial z}\left(\sigma_{zz}\frac{\partial u}{\partial z}+\sigma_{zx}\frac{\partial u}{\partial x}+\sigma_{zy}\frac{\partial u}{\partial y}\right)=\varphi(x,y,z).$$

To discretize this equation we will use finite difference approximation of the spatial derivatives on the reactangular grid (x_i, y_j, z_k), $i=\overline{1,N_x}$, $j=\overline{1,N_y}$, $k=\overline{1,N_z}$, where N_x, N_y, N_z are the numbers of grid points in *x,y, z* spatial directions. The finite difference approximation of the second order accuracy for the Poisson equation with mixed derivatives can be made with a minimal stencil of 7 points in 2D [9]. Generalization to 3D leads to a 13-point stencil, as shown in Fig. 1. It consists of two diagonal compartments (cells) with one common corner. The whole problem computational domain is represented by a 3D checkerboard lending itself for domain decomposition (partitioning). One can take into account only even (or only odd) mesh cells, each of them having eight neighboring computational cells. Every internal node of this checkerboard grid belongs simultaneously to two neighboring cells. Therefore, it is natural to introduce two components of an approximate numerical solution, (u_m, $\bar{u}_{9-m}$), where $m=\overline{1,8}$ (see Fig. 1). The first component of such pair, u_m, is considered as an internal component of the given mesh cell while the second one is a complimentary component belonging to the corresponding neighboring mesh cell. In these notations, the finite difference approximation, *L*, of the differential operator in (1) in an arbitrary node of the grid, (x_i, y_j, z_k), can be represented as

$$Lu = A_m u + \bar{A}_m \bar{u} \ , \qquad (3)$$

where $u = (u_1, u_2, \dots, u_8)^T$ and $\bar{u} = (\bar{u}_8, \bar{u}_7, \dots, \bar{u}_1)^T$ are the vectors of two components of the approximate numerical solution in two neighboring cells on the grid with a common node at (x_i, y_j, z_k).

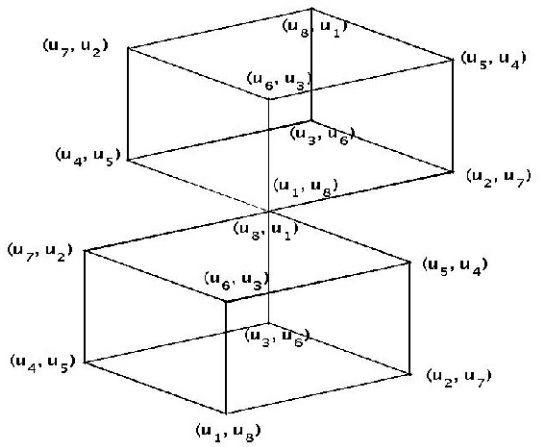


Fig. 1. Schematic view of the finite difference stencil for (1)

In (3) factors A_m and $\bar{A}_m$ are vectors with components given by coefficients of the finite difference approximation for (1), which is obtained by the standard finite volume method [10]. As a result, the derivatives in (1) are given by the following finite differences:

$$
\begin{aligned}
&\frac{\partial}{\partial x}\left(\sigma_{xx}\frac{\partial u}{\partial x}\right) \cong \sigma_{xx}^{12}\frac{u_2 - u_1}{h_x^2/2} - \sigma_{xx}^{78}\frac{\bar{u}_8 - \bar{u}_7}{h_x^2/2}, \quad \frac{\partial}{\partial y}\left(\sigma_{yy}\frac{\partial u}{\partial y}\right) \cong \sigma_{yy}^{14}\frac{u_4 - u_1}{h_y^2/2} - \sigma_{yy}^{58}\frac{\bar{u}_8 - \bar{u}_5}{h_y^2/2},\\
&\frac{\partial}{\partial z}\left(\sigma_{zz}\frac{\partial u}{\partial z}\right) \cong \sigma_{zz}^{16}\frac{u_6 - u_1}{h_z^2/2} - \sigma_{zz}^{38}\frac{\bar{u}_8 - \bar{u}_3}{h_z^2/2},\\
&\frac{\partial}{\partial x}\left(\sigma_{xy}\frac{\partial u}{\partial y}\right) \cong \sigma_{xy}^{34}\frac{u_3 - u_4}{2h_x h_y} - \sigma_{xy}^{12}\frac{u_2 - u_1}{2h_x h_y} + \sigma_{xy}^{78}\frac{\bar{u}_8 - \bar{u}_7}{2h_x h_y} - \sigma_{xy}^{56}\frac{\bar{u}_5 - \bar{u}_6}{2h_x h_y},\\
&\frac{\partial}{\partial y}\left(\sigma_{yx}\frac{\partial u}{\partial x}\right) \cong \sigma_{yx}^{23}\frac{u_3 - u_2}{2h_x h_y} - \sigma_{yx}^{14}\frac{u_4 - u_1}{2h_x h_y} + \sigma_{yx}^{58}\frac{\bar{u}_8 - \bar{u}_5}{2h_x h_y} - \sigma_{yx}^{76}\frac{\bar{u}_7 - \bar{u}_6}{2h_x h_y},\\
&\frac{\partial}{\partial x}\left(\sigma_{xz}\frac{\partial u}{\partial z}\right) \cong \sigma_{xz}^{25}\frac{u_3 - u_4}{2h_x h_z} - \sigma_{xz}^{16}\frac{u_6 - u_1}{2h_x h_z} + \sigma_{xz}^{38}\frac{\bar{u}_8 - \bar{u}_3}{2h_x h_z} - \sigma_{xz}^{47}\frac{\bar{u}_7 - \bar{u}_4}{2h_x h_z},\\
&\frac{\partial}{\partial z}\left(\sigma_{zx}\frac{\partial u}{\partial x}\right) \cong \sigma_{zx}^{56}\frac{u_5 - u_6}{2h_x h_z} - \sigma_{zx}^{12}\frac{u_2 - u_1}{2h_x h_z} + \sigma_{zx}^{78}\frac{\bar{u}_8 - \bar{u}_7}{2h_x h_z} - \sigma_{xz}^{34}\frac{\bar{u}_3 - \bar{u}_4}{2h_x h_z},
\end{aligned}
\tag{4}
$$

$$\frac{\partial}{\partial y}\left(\sigma_{yz}\frac{\partial u}{\partial z}\right)\cong\sigma_{yz}^{47}\frac{u_7-u_4}{2h_yh_z}-\sigma_{yz}^{16}\frac{u_6-u_1}{2h_yh_z}+\sigma_{yz}^{38}\frac{\bar{u}_8-\bar{u}_3}{2h_yh_z}-\sigma_{yz}^{25}\frac{\bar{u}_5-\bar{u}_2}{2h_yh_z},$$

$$\frac{\partial}{\partial z}\left(\sigma_{zy}\frac{\partial u}{\partial y}\right)\cong\sigma_{zy}^{67}\frac{u_7-u_6}{2h_yh_z}-\sigma_{zy}^{14}\frac{u_4-u_1}{2h_yh_z}+\sigma_{zy}^{58}\frac{\bar{u}_8-\bar{u}_5}{2h_yh_z}-\sigma_{zy}^{23}\frac{\bar{u}_3-\bar{u}_2}{2h_yh_z}.$$

Here $\sigma^{mk}=2\sigma^m\sigma^k/(\sigma^m+\sigma^k)$, where σ^k,σ^m are values of the conductivity tensor components in nodes k and m, and h_x,h_y,h_z are grid steps along the Cartesian axis. As it is seen from (4), variables u_1 and $\bar{u}_8$, which correspond to the most distant nodes in the two cell arrangement in Fig. 1, are absent. This means these nodes are not involved into the stencil. By grouping the terms belonging to one of two cells in the stencil in expressions for finite difference derivatives in (4) one can obtain an additive representation of operator L in (3), which allows us to express the components of vectors A_m and $\bar{A}_m$. For instance, for A_1 and $\bar{A}_8$ we have:

$$A_1u=-\left(\frac{\sigma_{xx}^{12}}{h_x^2}+\frac{\sigma_{yy}^{14}}{h_y^2}+\frac{\sigma_{zz}^{16}}{h_z^2}-\frac{\sigma_{xy}^{14}+\sigma_{yx}^{12}}{2h_xh_y}-\frac{\sigma_{xz}^{16}+\sigma_{zx}^{12}}{2h_xh_z}-\frac{\sigma_{yz}^{16}+\sigma_{zy}^{14}}{2h_yh_z}\right)u_1+$$

$$+\left(\frac{\sigma_{xx}^{12}}{h_x^2}-\frac{\sigma_{xy}^{32}+\sigma_{yx}^{12}}{2h_xh_y}-\frac{\sigma_{xz}^{25}+\sigma_{zx}^{12}}{2h_xh_z}\right)u_2+\frac{\sigma_{xy}^{32}+\sigma_{yx}^{34}}{2h_xh_y}u_3+\left(\frac{\sigma_{yy}^{14}}{h_y^2}-\frac{\sigma_{xy}^{14}+\sigma_{yx}^{34}}{2h_xh_y}-\frac{\sigma_{yz}^{47}+\sigma_{zy}^{14}}{2h_yh_z}\right)u_4+$$

$$+\frac{\sigma_{xz}^{25}+\sigma_{zx}^{56}}{2h_xh_z}u_5-\frac{\sigma_{xz}^{16}+\sigma_{zx}^{56}}{2h_xh_z}u_6+\frac{\sigma_{yz}^{47}+\sigma_{zy}^{67}}{2h_yh_z}u_7;\ A_8u=\frac{\sigma_{yz}^{25}+\sigma_{zy}^{23}}{2h_yh_z}u_2+\left(\frac{\sigma_{zz}^{38}}{h_z^2}-\frac{\sigma_{xz}^{38}+\sigma_{zx}^{34}}{2h_xh_z}\right)u_3$$

$$+\frac{\sigma_{xz}^{47}+\sigma_{zx}^{34}}{2h_xh_z}u_4+\left(\frac{\sigma_{yy}^{58}}{h_y^2}-\frac{\sigma_{xy}^{58}+\sigma_{yx}^{65}}{2h_xh_y}\right)u_5++\frac{\sigma_{xy}^{67}+\sigma_{yx}^{65}}{2h_xh_y}u_6+\left(\frac{\sigma_{xx}^{78}}{h_x^2}-\frac{\sigma_{xy}^{67}+\sigma_{yx}^{87}}{2h_xh_y}-\frac{\sigma_{xz}^{47}+\sigma_{zx}^{78}}{2h_xh_z}\right)u_7-$$

$$-\left(\frac{\sigma_{xx}^{78}}{h_x^2}+\frac{\sigma_{yy}^{58}}{h_y^2}+\frac{\sigma_{zz}^{38}}{h_z^2}-\frac{\sigma_{xy}^{58}+\sigma_{yx}^{87}}{2h_xh_y}-\frac{\sigma_{xz}^{38}+\sigma_{zx}^{78}}{2h_xh_z}-\frac{\sigma_{yz}^{38}+\sigma_{zy}^{58}}{2h_yh_z}\right)u_8.$$

The similar expressions are obtained for three remaining pairs of operators A_2 and A_7, A_3 and A_6, A_4 and A_5. In the boundary voxels of the computational domain the finite difference approximation is constructed taking into account the boundary conditions.

In the particular case of identity between two complimentary components $u_m\equiv\bar{u}_{m'}$, the numerical scheme presented above is equivalent to a system of finite difference equations with a 13 diagonal matrix and dimension $N=N_x\times N_y\times N_z$, where N is a total number of nodes in the grid. The high dimensionality of a finite difference model is a major obstacle in the computational complexity of this numerical problem. The introduction of additional (complimentary) solution components opens an opportunity for use of the vector-additive iterative methods [7-9], which are unconditionally stable and potentially amenable for multi-threading limited only by a total number of nodes in a grid.

An application of the vector-additive iterative scheme of the domain decomposition type to our problem leads to an algorithm with the following key features. Iterative approximations for the internal components in every cell of the grid are computed implicitly as solutions of the system of eight linear algebraic equations in respect of these unknown internal components. External components (belonging to eight neighboring cells) in such an implicit solution are taken from the previous iteration step. As a result, an elementary per-voxel step of the iterative process consists of solving a system of linear algebraic equations of the following type:

$$\frac{\overset{k+1}{u}_m-\tilde{u}}{\tau}=\lambda A_m(\overset{k+1}{u}-\overset{k}{u})+A_m\overset{k}{u}+\bar{A}_m\overset{k}{\underline{u}}+\varphi,\ m=\overline{1,8},\ \tilde{u}=(u_m+\bar{u}_{9-m})/2\ . \tag{5}$$

Here, iteration parameters $\tau > 0$ and $\lambda \geq 1$, where k is an iteration number. Apparently, the calculation of the next iterative approximation requires solving a system of 8 equations of type (5). Thus, the computational complexity per iteration is $Q=NQ_0/8$, where Q_0 is the computational cost for solving the linear system in (5) with a matrix 8×8, and $N/8$ is a number of computational cells in the checkerboard discretization. Assuming the Gaussian elimination algorithm for solving (5), we have approximately $Q_0 \sim (2/3)8^3 \approx 341$ floating operations per–cell at one iteration. Thus, the computational complexity per iteration is comparable with the standard PCG algorithms. The most important point is that an iterative solution in every computational cell can be updated concurrently as it is dependent from the neighboring cells input only from the previous iteration. Therefore, the structure of this algorithm allows natural partitioning up to N/8 parallel threads of execution.

Theoretical estimates of convergence for this class of the vector-additive numerical schemes and optimal choice of iteration parameters have been developed by Abrashin et. al. [7,8]. An example of using the similar iterative scheme in a 2D case for the convection-diffusion equation was given in one of our work [9].

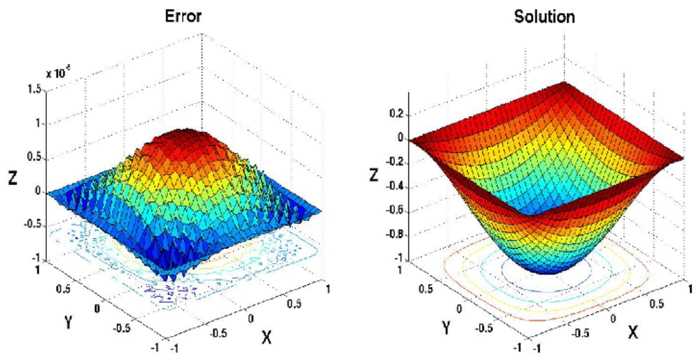


Fig. 2. Local error (left) and numerical solution (right) for a test analytical case (see text for details)

4 Validation and Numerical Examples

A serial version of the proposed forward anisotropic solver was prototyped in Matlab. It was validated against an analytical solution and tested on a cubic phantom with anisotropic inclusions.

A simple analytic test was constructed assuming that in a cubic computational domain with edge length 2a the solution has the form:

$$u(x,y,z) = (x-a)(x+a)(y-a)(y+a)(z-a)(z+a).$$

Apparently, such a solution satisfies the Dirichlet boundary conditions at the computational domain boundaries. The right-hand term, $\varphi(x, y, z)$, has been found by direct analytical differentiation of $u(x, y, z)$ according to (1) and a set of analytical conductivity tensor components. In Fig. 2 one can see the good agreement between the analytical and numerical solutions. The error between analytical and numerical solutions was computed in terms of the local norm. The algorithm converged at 54 iterations with accuracy 1.e-6 for the problem size 32x32x32 voxels. In addition, we

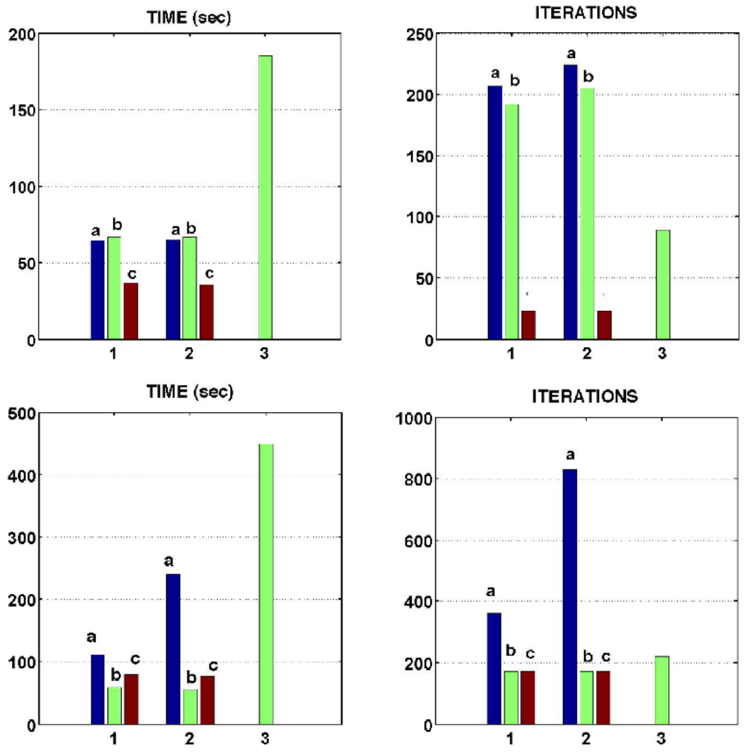


Fig. 3. Histograms of computational time (left) and number of iterations (right) to convergence for QMR (1), BiCG (2) and vector-additive method (3). Preconditioning: without (a), Jacobi (b) and IChF (c). Coefficients and accuracy: smooth, 1.e-6 (top) and heterogeneous, 1.e-4 (bottom).

compared performance of the vector-additive algorithm against the Quasi-Minimal Residual method (QMR) and BiConjugate Gradients method (BiCG) constructed with the same 13-point stencil, as in Fig. 1, and with different preconditioners (Jacobi and incomplete Cholesky factorization (IChF)) for smooth and highly heterogeneous anisotropic phantoms. The code for QMR and BiCG was prototyped in Matlab using the classic schemes [11-13].

As seen in Fig. 3, the QMR and BiCG algorithms perform about 4-5 times better than the vector-additive algorithm in terms of computational time for the heterogeneous and smooth problems of size 64x64x64. This is not surprising, as the serial vector-additive algorithm in the present Matlab implementation is not optimized in terms of matrix operations, while the QMR and BiCG implementations are completely vectorized by using the standard Matlab functions. Yet, the convergence of the vector-additive algorithm was found to be comparable (Fig. 3, right) in terms of a number of iterations needed to reach the prescribed accuracy.

For simulation of the more realistic case of the human head geometry we have employed a cubic phantom with the 20 centimeters edge. The phantom has several shells representing air, scalp, skull, Cerebro-Spinal Fluid (CSF) and different anisotropic inclusions modeling brain. The isotropic conductivity values of scalp (0.45 S/m), skull (0.018 S/m), CSF (1.9 S/m) have been chosen to be equal to the median values reported in the published literature [6]. The air conductivity has been set to 0.001 S/m. The anisotropic ratio of conductivity in the brain inclusion has been set to 1:10 in the orthotropic directions. The results of the current streamline calculations generated by a source and a sink placed in different anisotropic parts of brain and convergence of the vector-additive method and the BiCG method versus the number of discretization points along one direction are shown in Fig.4. Again, in terms of the number of iterations, K_ε, the vector-additive algorithm performance is comparable with the BiCG method. One can see that both methods are converging at the rate of about 300 iterations for the problem size 100x100x100. It is worth noting, that the Jacobi preconditioner performed much better in the case of heterogeneous anisotropic inclusions (comparable with performance of the IChF preconditioner), while in the case of the homogeneous anisotropic cube (Fig. 3, the top-right corner)

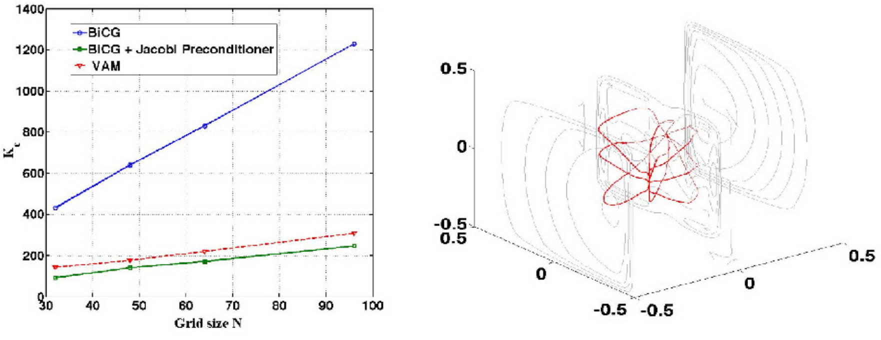


Fig. 4. Anisotropic phantom simulation. Left: convergence of the BiCG(with and without the Jacobi preconditioner), and vector-additive method (the red line) versus the grid size. Right: the current streamlines inside the brain phantom.

the BiCG method with the IChF preconditioner converged essentially faster. The current streamlines shown in Fig.4, right by the thick red color lines behave as expected in accordance with the anisotropic ratio model chosen for the brain inclusions: preferably vertically for the source, horizontally for the sink and equidistant in the surrounding isotropic CSF.

5 Conclusion

We have described a novel 3D finite volume algorithm for solving the anisotropic heterogeneous Poisson equation based on the vector-additive implicit methods with a 13-point stencil. The proposed multi-component additive algorithm is unconditionally stable in 3D and amenable for domain decomposition parallelization with a high number of threads, limited only by the number of grid points in the initial computational domain. We have introduced two major modifications to the classic multi-component vector-additive method suggested in [7-9]. First, we have reduced the number of components from four to two in 3D by using the checkerboard discretization which relaxes the requirements for the operational memory. In the original version of this method [7,8] the minimal number of components was estimated to be $2^{(D-1)}$, where D is the dimension of a computational problem. Secondly, we have introduced variable iterative parameters to improve the convergence rate in the case of essentially heterogeneous coefficients. Finally, to the best of our knowledge, this is the first attempt to use the multi-component numerical scheme for solving 3D anisotropic problems.

The estimated computational complexity per iteration and the method serial performance are found to be comparable to the most efficient iterative solvers from the family of the preconditioned conjugate gradient (PCG) algorithms, in particular the BiCG method with the Jacobi and IChF preconditioners. In the present Matlab implementation the serial version takes more time per iteration and to converge than the standard methods due to the specifics of Matlab, where the PCG algorithms are completely vectorized, while our method can not avoid some necessary cycles. We expect the serial performance to be significantly better in the case of C/C++ implementation. We believe the 3D vector additive method has better parallelism potential than PCG methods due to its cell-level data decomposition. We expect to see performance improvements that overcome the sequential deficiencies as the resolution of the head model scales. Our next step will be a parallel implementation of this algorithm on a computational cluster and a GPGPU accelerator for large size problems based on the high-resolution (256x256x256 voxels) human MRI/CT data.

References

1. Arridge, S.R.: Optical Tomography in Medical Imaging. Inverse Problems 15, R41–R93 (1999)
2. Gulrajani, R.M.: Bioelectricity and Biomagnetism. John Wiley & Sons, New York (1998)
3. Hallez, H., et al.: Review on Solving the Forward Problem in EEG Source Analysis. Journal of Neuroengineering and Rehabilitation 4, 46 (2007)
4. General-Purpose Computation Using Graphics Hardware, `http://www.gpgpu.org`

5. Qin, C., Kang, N., Cao, N.: Performance evaluation of anisotropic diffusion simulation based tractography on phantom images. In: 45th Annual Southeast Regional Conference ACMSE 2007, pp. 521–522. ACM, New York (2007)
6. Salman, A., Turovets, S., Malony, A., Volkov, V.: Multi-Cluster, Mix-Mode Computational Modeling of Human Head Conductivity. In: Mueller, M.S., et al. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 119–130. Springer, Heidelberg (2008)
7. Abrashin, V.N., Egorov, A.A., Zhadaeva, N.G.: On the Convergence Rate of Additive Iterative Methods. Differential Equations 37, 867–879 (2001)
8. Abrashin, V.N., Egorov, A.A., Zhadaeva, N.G.: On a Class of Additive Iterative Methods. Differential Equations 37, 1751–1760 (2001)
9. Volkov, V.M., Lechtikov, S.N.: Multicomponent Iterative Methods of Decomposition Type for Two-Dimensional Stationary Problems of Dissipative Transfer. Differential Equations 33, 927–933 (1997)
10. LeVeque, R.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge (2002)
11. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: The Numerical Recipes in C: The Art of Scientific Computing, 2nd edn. Cambridge University Press, New York (1992)
12. Barrett, R., Berry, M., Chan, T.F., et al.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, Philadelphia (1994)
13. Freund, R., Nachtigal, N.: QMR: A Quasi-Minimal Residual Method for Non-Hermitian Linear Systems. Numer. Math. 60, 315–339 (1991)

Hash Functions Based on Large Quasigroups

Václav Snášel[1], Ajith Abraham[2], Jiří Dvorský[1], Pavel Krömer[1], and Jan Platoš[1]

[1] Department of Computer Science, FEECS, VŠB – Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{vaclav.snasel}@vsb.cz

[2] Machine Intelligence Research Labs (MIR Labs), Auburn, Washington 98071, USA
ajith.abraham@ieee.org

Abstract. In this article we discuss a simple hash function based upon properties of a well-known combinatorial design called quasigroups. The quasigroups are equivalent to the more familiar Latin squares and one of their most important properties is that all possible element of certain quasigroup occurs with equal probability. Actual implementations are based on look-up table implementation of the quasigroup, which is unusable for large quasigroups. In contrast, presneted hash function can be easily implemented. It allows us to compute hash function without storing large amount of data (look-up table). The hash function computation is illustrated by experiments summarized in the last section of this paper.

1 Introduction

The need for random and pseudorandom sequences arises in many applications, e.g. in modelling, simulations, and of course in cryptography. Pseudorandom sequences are the core of stream ciphers. They are popular due to their high encryption/decryption speed. Their simple and cheap hardware design is often preferred in real-world applications. The design goal in stream ciphers is to efficiently produce pseudorandom sequences - keystreams (i.e. sequences that possess properties common to truly random sequences and in some sense are "indistinguishable" from these sequences).

Hash functions map a large collection of messages into a small set of message digests and can be used for error detection, by appending the digest to the message during the transmission (the appended digest bits are also called parity bits). The error will be detected if the digest of the received message, in the receiving end, is not equal to the received message digest. This application of hash functions is only for random errors, since an active spoofer may intercept the transmitted message, modify it as he wishes, and resend it appended with the digest recalculated for the modified message.

With the advent of public key cryptography and digital signature schemes, cryptographic hash functions gained much more prominence. Using hash functions, it is possible to produce a fixed length digital signature that depends on the whole message and ensures authenticity of the message. To produce digital signature for a message M, the digest of M, given by $H(M)$, is calculated and then encrypted with the secret key of the sender. Encryption may be either by using a public key or a private key algorithm. Encryption of the digest prevents active intruders from modifying the message and recalculating its check sum accordingly. It effectively divides the universe into two groups: outsiders who do not have access to the key of the encryption algorithm and hence cannot effectively produce a valid checksum, and insiders who do have access to the key and hence can produce valid checksums. We note that in a public key algorithm, the group of insiders consists of only one member (the owner of the private key) and hence the encrypted hash value uniquely identifies the signer. In the case of symmetric key algorithms, both the transmitter and the receiver have access to the secret key and can produce a valid encrypted hash for an arbitrary message and therefore, unique identification based on the encrypted hash is not possible. However, an outsider cannot alter the message or the digest.

In the study of hash functions, Information Theory and Complexity Theory are two major approaches. The methods based on information theory provide unconditional security — an enemy cannot attack such systems even if he/she has unlimited power. This approach is generally impractical.

In the second approach, some assumptions are made based on the computing power of the enemy or the weaknesses of the existing systems and algorithms, and therefore, the security cannot be proven but estimated by the analysis of the best known attacking algorithms and considering the improvements of the hardware and softwares. In other words, hash functions based on complexity theory are computationally secure. In this paper, we concentrate on the second approach.

1.1 Definitions

Definition 1. *A function $H()$ that maps an arbitrary length message M to a fixed length hash value $H(M)$ is a OneWay Hash Function (OWHF), if it satisfies the following properties:*

1. *The description of $H()$ is publicly known and should not require any secret information for its operation.*
2. *Given M, it is easy to compute $H(M)$.*
3. *Given $H(M)$ in the rang of $H()$, it is hard to find a message M for given $H(M)$, and given M and $H(M)$, it is hard to find a message $M'(\neq M)$ such that $H(M') = H(M)$.*

Definition 2. *A function $H()$ that maps an arbitrary length message M to a fixed length hash value is a Collision Free Hash Function (CFHF), if it satisfies the following properties:*

1. *The description of $H()$ is publicly known and should not require any secret information for its operation.*
2. *Given M, it is easy to compute $H(M)$.*
3. *Given $H(M)$ in the rang of $H()$, it is hard to find a message M for given $H(M)$, and given M and $H(M)$, it is hard to find a message $M'(\neq M)$ such that $H(M') = H(M)$.*
4. *It is hard to find two distinct messages M and M' that hash to the same result ($H(M) = H(M')$).*

2 Construction of Hashing Function Based on Quasigroup

Definition 3. *Let Q be a nonempty set with one binary operation $(*)$. Then Q is said to be a grupoid and is denoted by $(Q, *)$.*

Definition 4. *A grupoid $(Q, *)$ is said to be a quasigroup (i.e. algebra with one binary operation $(*)$ on the set Q) satisfying the law:*

$$(\forall u, v \in Q)(\exists! x, y \in Q)(u * x = v \wedge y * u = v).$$

This implies:

1. $x * y = x * z \vee y * x = z * x \Rightarrow y = z$
2. The equations $a * x = b, y * a = b$ have an unique solutions x, y for each $a, b \in Q$.

However, in general, the operation (*) is neither a commutative nor an associative operation.

Quasigroups are equivalent to the more familiar Latin squares. The multiplication table of a quasigroup of order q is a Latin square of order q, and conversely, as it was indicated in [1,2,8], every Latin square of order q is the multiplication table of a quasigroup of order q.

Definition 5. *Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite alphabet, a $k \times n$ Latin rectangle is a matrix with entries $a_{ij} \in A$, $i = 1, 2, \ldots, k$, $j = 1, 2, \ldots, n$, such that each row and each column consists of different elements of A. If $k = n$ we say a Latin square instead of a Latin rectangle. Latin square is called reduced (or in standard form) if both the first row and the left column are in some standard order, alphabetical order being convenient.*

All reduced Latin squares of order n are enumerated for $n \leq 10$ as it is shown in [3]. Let L_n be the number of Latin squares of order n, and let R_n be the number of reduced Latin squares of order n. It is easy to see that $L_n = n!(n-1)!R_n$. The problem of classification and exact enumeration of quasigroups of order greater than 10 probably still remains unsolved. Thus, there are more then 10^{90} quasigroups of order 16 and if we take an alphabet $A = \{0 \ldots 255\}$ (i.e. data are represented by 8 bits) there are at least $256!255! \ldots 2! > 10^{58000}$ quasigroups.

Multiplication in quasigroups has important property: It is proved that each element occurs exactly q times among the products of two elements of Q, q^2 times among the products of three elements of Q and, generally q^{t-1} among the products of t elements of Q. Since there are q^t possible ordered products of t elements of Q, this shows that each element occurs equally often among these q^t products (see [4]).

Definition 6. *Let $H_Q() : Q \rightarrow Q$ be projection defined as*

$$H_Q(q_1 q_2 \ldots q_n) = ((\ldots(a * q_1) * q_2 * \ldots) * q_n$$

Then $H_Q()$ is said to be hash function over quasigroup $(Q, *)$. The element a is a fixed element from Q.

Example 1. Quasigroup of modular subtraction has following table representation:

0	3	2	1
1	0	3	2
2	1	0	3
3	2	1	0

The table above defines quasigroup because it satisfies conditions to be Latin Square. Multiplication in the quasigroup is defined in following manner: $a * b = (a + 4 - b) \bmod 4$. It is obvious that the quasigroup is neither commutative ($1*2 = 3, 2*1 = 1$) nor associative. Value of hash function is $H_2(0013) = (((2 * 0) * 0) * 1) * 3 = 2$.

2.1 Sketch of Proof of Resistance to Attacks

Hash function based on quasigroup is iterative process which computes hash value (digest) for message $X = x_1 x_2 \ldots x_n$. Suppose that $H_Q(X) = d$. Hash function is preimage resistant when it is "impossible" to compute from given digest source message X. The digest d should be factorized into message $Y = y_1 y_2 \ldots y_n$. In the first step we can divide digest d into two parts y_1 and α_1, where $d = y_1 * \alpha_1$. In the second step value α_1 needs to be divided into y_2 and α_2 ($\alpha_1 = y_2 * \alpha_2$) and so for each element $y_i, 1 \leq i \leq n$. Because each y_i has a same probability of occurrence among products of Q, $|Q|^n$ possible choices should be checked to obtain message Y.

Definition 7. *Quasigroups Q and R are said to be homotopic, if there are permutations satisfying the law:* $(\forall u, v \in R)(u * v = \pi(\omega(u) * \rho(v)))$.

We can imagine homotopy of quasigroups as permutation of rows and columns of quasigroup's multiplication table.

Example 2. Table of quasigroup, which is homotopic with quasigroup of modular subtraction:

0	3	2	1
2	1	0	3
1	0	3	2
3	2	1	0

The table was created from table of modular subtraction. The second and the third row were exchanged. Permutations π, ρ are identities and $\omega = [0213]$. For example $1 * 0 = \omega(1) * 0 = 2 * 0 = 2$.

This example can be considered as a method how to construct new quasigroups. In following text we will use quasigroups homotopic with quasigroup of modular subtraction. Three random permutations will be generated and table will be used to modify original table. Such quasigroup we call "table quasigroup". Disadvantage of this method is huge space complexity (n^2 elements must be stored).

Homotopy gives us possibility to compute result of multiplication without table. Three functions must be chosen to calculate permutations π, ρ, ω. Then the multiplication is defined as follows: $a * b = \pi((\omega(a) + n - \rho(b)) \bmod n)$.

A sequence of n elements were divided into several parts; these were rotated in various directions and exchanged among themselves. Hereafter presented function P1 compute one of these permutations i.e. $P1(x) = \omega(x)$. Two other permutations are implemented in the same way.

```
const unsigned int cQuasiGroupA2::P1(unsigned int x) const
{
   unsigned int Dimension2 = m_Dimension / 2;
   if (x < Dimension2 * 2)
   {
      if (x & 1)
         x = 2 * ((x / 2 + 1) % Dimension2) + 1;
      else
         x = 2 * ((x / 2 + Dimension2 - 1) % Dimension2);
   }
   return x;
}
```

This enables us to work with large quasigroups. Works that are already known use quasigroups of small order only, or only a small parts of certain quasigroup are there used mainly as a key for Message Authentication Code [3]. These are represented as a look-up table in main memory. Hypothesis mentioned above will be tested in next section.

3 Experimental Results

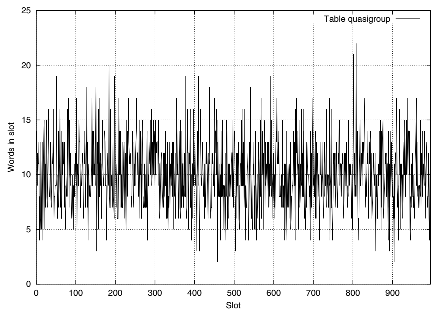
A simple application was created to verify our hypothesis and expectations. Inputs to the application were sets of distinct words, which were extracted from particular text file. The first input was file *bible* from Canterbury Corpus [5] (section Large files) and it was about 4 megabytes long. About 10000 distinct words were extracted from this file. The second was file *latimes* from TREC document corpus (see `http://trec.nist.gov`). The file contained Los Angeles Times volumes 1989 and 1990. And it was about 450 megabytes long. We extracted 200000 distinct words from this text file.

To get statistical data about distribution of words in range of hash values and other properties imaginary hash table have been implemented and values in the table were measured. We observed several parameters:

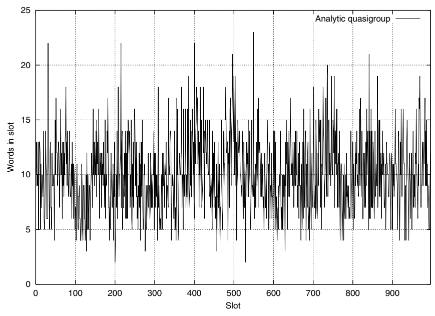
1. divergences in numbers of words in slots for given size of hash table between our hash function and uniform distribution of words in table,
2. distribution of lengths of slots,
3. how many bits are changed, when one bit in input has been inverted,
4. probability of bits alternation in given position, when one bit in input has been inverted.

3.1 Distribution of Words in Slots

The distribution of words in slots of hash table is figured in charts 1, 2. It can be seen from charts 1(a) and 1(b) that distribution of words in slots of imaginary hash table is quite uniform, both for table quasigroup and for analytic one. But in chart 2 there are differences between table and analytic quaisgroup. Moreover analytic results have regular shape. This error is caused by constant parameters of functions that compute permutations in analytic quasigroup.

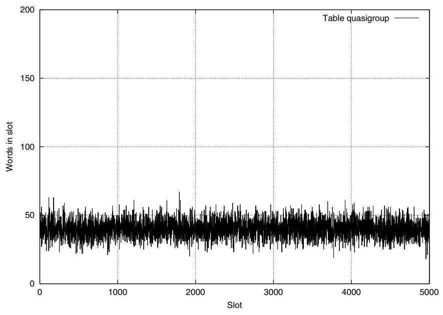


(a) Table quasigroup

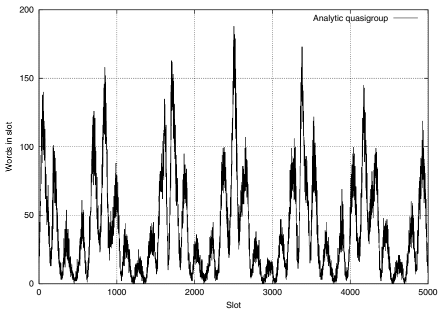


(b) Analytic quasigroup

Fig. 1. Distribution of words in slots for file *bible*, hash size 997



(a) Table quasigroup



(b) Analytic quasigroup

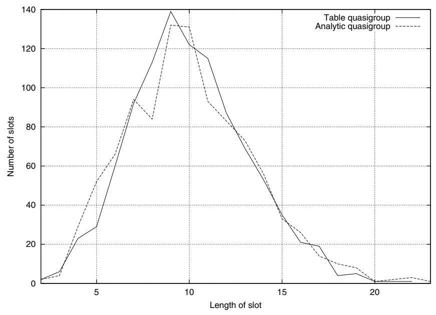
Fig. 2. Distribution of words in slots for file *latimes*, hash size 5003

3.2 Distribution of Lengths of Slots

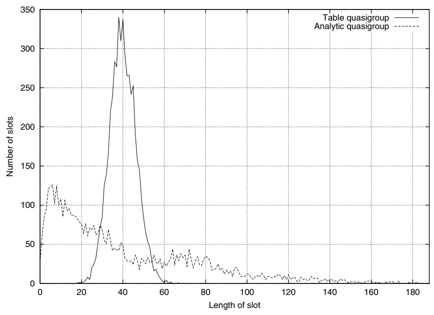
We can observe good correspondence between distribution of table quasigroup and analytic quasigroup in chart 3(a) for the file *bible*. For file *latimes* there is absolute divergence in chart 3(b).

3.3 Probability of Change of Particular Number of Bits

We focus on influence of inputs change on resultant value of hash function. Step by step every bit in each input word was inverted and value of hash function was computed.

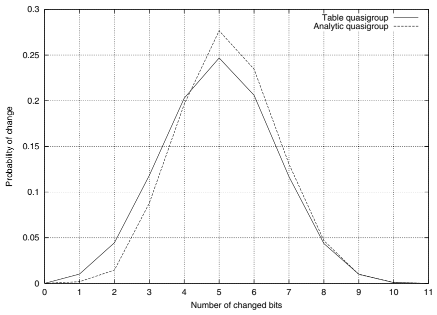


(a) *bible*, hash size 997

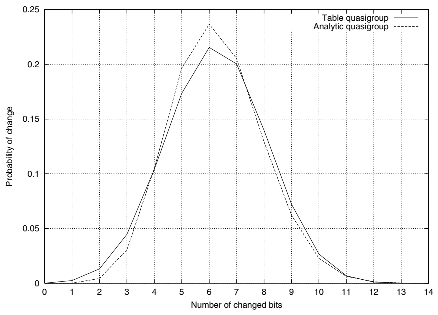


(b) *latimes*, hash size 5003

Fig. 3. Distribution of slots' length



(a) *bible*, hash size 997



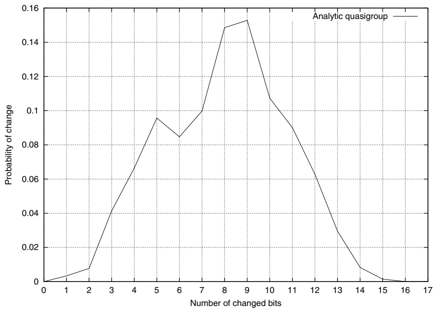
(b) *latimes*, hash size 5003

Fig. 4. Probability of change of particular number of bits

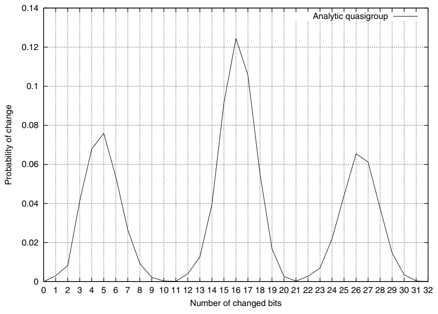
Then Hamming distance between hash values for original word and modified one was measured. It can be seen from chart 4 that both distribution curve has the same shape and they are very close together. It is interesting especially for chart 4(b) with respect of bad characteristic of slots distribution in chart 3(b). Next we perform experiments with analytic quasigroup of order 2^{16} and 2^{32} i.e. for 16 and 32 bit long numbers. Result of the experiment is given in chart 5.

3.4 Probability of Bits Alternation in Given Position

Alternations of bits in specific positions in result of hash function were observed. The experiment runs with the same conditions as previous experiment, but we kept track to positions of changed bits. Only minor errors can be seen (chart 6) between table quasigroup and analytic quasigroup. The changes are uniformly distributed over all bits in resultant hash value.

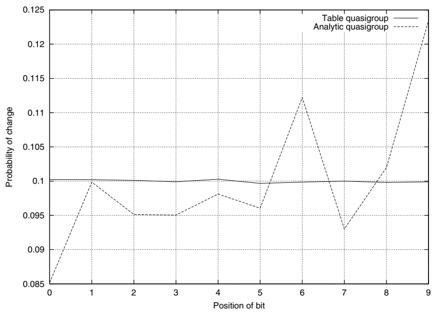


(a) *latimes*, hash size 2^{16}

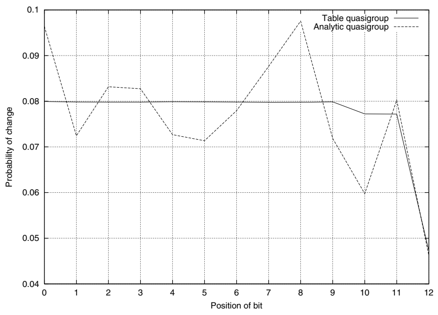


(b) *latimes*, hash size 2^{32}

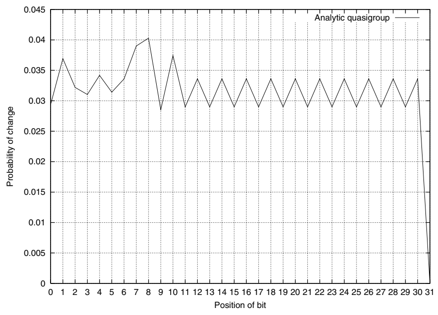
Fig. 5. Probability of change of particular number of bits



(a) *bible*, hash size 997



(b) *latimes*, hash size 5003



(c) *latimes*, hash size 2^{32}

Fig. 6. Probability of particular bit's change

4 Conclusion and Future Works

We presented hash function based on non-associative algebraic structures. This work is continuation of our paper [6]. The presented hash function can be easily implemented. Comparison between look-up table and analytic quasigroup implementation is given.

Analytic quasigroup has some faults in it properties, but there is no need to store large table. For real usage arithmetic of long numbers (i.g. 512 bits) must be adopted. Non-associative structure - neofield - could be incorporated in our future works.

References

1. Belousov, V.D.: Osnovi teorii kvazigrup i lup. Nauka, Moscow (1967) (in Russian)
2. Dénes, J., Keedwell, A.: Latin Squares and their Applications, Akadémiai Kiadó, Budapest. Academic Press, New York (1974)
3. McKay, B., Rogoyski, E.: Latin square of order 10. Electronic Journal of Combinatorics (1995), http://www.emis.de/journals/EJC/Volume_2/cover.html
4. Dénes, J., Keedwell, A.: A new authentication scheme based on latin squares. Discrete Mathematics (106/107), 157–161 (1992)
5. Arnold, R., Bell, T.: A corpus for evaluation of lossless compression algorithms. In: DCC 1997: Proceedings of the Conference on Data Compression, p. 201 (1997)
6. Dvorský, J., Ochodková, E., Snášel, V.: Hash Functions Based on Large Quasigroups. In: Proceedings of Velikonoční kryptologie, Brno, pp. 1–8 (2002)
7. Ochodková, E., Snášel, V.: Using Quasigroups for Secure Encoding of File System. In: Proceedings of the International Scientific NATO PfP/PWP Conference Security and Information Protection 2001, Brno, Czech Republic, pp. 175–181 (2001)
8. Smith, J.D.H.: An introduction to quasigroups and their representations. Chapman and Hall, Boca Raton (2007)

Second Derivative Approximation for Origin-Based Algorithm

Feng Li

IBM China Research Laboratory,
Beijing 100193, P.R.China
lfeng@cn.ibm.com

Abstract. Origin-based algorithm(OBA) for traffic assignment problem has been demonstrated preferable to the widely accepted and used Frank-Wolfe algorithm and path-based algorithms. Note that OBA can not avoid path enumeration of concerned network, which will lead to two disadvantages. One is the intensive memory requirements and the other is the difficulties in manipulating and storing paths. In order to solve these problems, we first propose the lower and upper bounds of the Hessian matrix, which can be calculated without path enumeration. Then use the lower bound of Hessian matrix to approximate the direction of the origin-based algorithm. According to our computational results, the modified origin-based algorithm(MOBA) improves the convergence performance greatly. The results indicate that MOBA can deliver better and more reliable convergence than OBA and saves much more CPU time especially when large-scale networks are being considered.

Keywords: Traffic Assignment, Origin-based Algorithm, Second Derivative, User Equilibrium.

1 Introduction

Traffic assignment problem (TAP) is the key problem for the long term planning and evaluation of urban transportation network. It is to assign the traffic flows of each OD (Origin-Destination) pair to links of urban transportation network in terms of certain principle, and count out all link traffic flows. There are many principles for TAP, but the most often used principles are the first and second principles of Wardrop, i.e. User Equilibrium principle and System Optimum principle [1]. In order to find a solution satisfying the User Equilibrium principle, Beckmann et al. proposed a convex mathematical programming [2][3] which became the main tool for solving traffic assignment problem. Since the work of Beckmann et al, many algorithms have been suggested to solve it. All of them can be broadly divided into three categories according to the solution space the algorithm resides, i.e., the link- path- or origin-based algorithms.

The link-based algorithms, including Frank-Wolfe (FW) algorithm given by LeBlanc et al. [4] and several modified FW algorithms [5][6][7][8][9], yield link-based solutions. The path-based algorithms, including disaggregated simplicial decomposition (DSD) algorithm [10], gradient projection (GP) algorithm [11][12] and conjugate gradient projection (CGP) algorithm [13] etc., can determine both an aggregate link-based solution and an individual path-based solution which is not available for link-based algorithms. The origin-based algorithm(OBA)[14] can provide both an aggregate link-based solution and a constructive path-based solution. In the past, path-based algorithms and origin-based algorithm were not considered as a viable solution approach for large-scale network problems because of intensive memory requirements and difficulties in manipulating and storing paths. However, path-based algorithms and origin-based algorithm are now made possible due to the dramatic advances in computing technology.

Recent researches [13][14] have shown and demonstrated that the origin-based algorithm excels not only the FW algorithm but also some path-based algorithms such as DSD, GP and CGP. Although OBA was designed for solutions with high levels of accuracy, it also can not avoid manipulating and storing paths in each iteration of the algorithm. Plentiful computational experiments of OBA also indicate that most of the CPU time of the OBA is to enumerate and manipulate paths of the restricting subnetwork $A_p(\alpha)$(see [14]).

Motivated by the problems discussed above, we give a second derivative approximation to modify the origin-based algorithm. By using this approximation, the modified origin-based algorithm (MOBA) can avoid enumerating and manipulating paths of the restricting subnetworks and outperforms OBA in the speed of convergence and memory requirements. The remaining sections of this paper are organized as follows. Section 2 describes the traffic assignment problem and introduces the origin-based algorithm. The modified origin-based algorithm by using second derivative approximation is then proposed in section 3. Section 4 describes the computational results generated by OBA and MOBA from two real networks including Sioux Falls and Barcelona network. Finally, a brief conclusion is given in section 5.

2 User-Equilibrium Traffic Assignment and Origin-Based Algorithm

Before introducing the user equilibrium traffic assignment model and the origin-based algorithm, we give the following notations which are used in this paper.

G	$G = (\mathcal{N}, \mathcal{A})$ represent an urban transportation network
$\mathcal{N}$	$\mathcal{N} = (1,2,\cdots,N)$ the set of nodes
$\mathcal{A}$	the set of all directed links
O	the set of origin nodes
D	the set of destination nodes
a	$a = [a_t, a_h]$ the directed link of the network
a_t	the tail node of link a
a_h	the head node of link a
x_a	the traffic flow on link a
c_a	the traffic cost on link a

c'_a	the first-order derivative of the cost of link a
x_a^p	the traffic flow on link a from origin p
α_a^p	the proportion of the traffic flow from origin p that arrived at a_h through link a
μ_a^p	the average cost of link a for origin p
σ_i^p	the average cost from origin p to node i
ν_a^p	the average first-order derivative of the cost of link a for origin p
ρ_i^p	the average first-order derivative of the cost from origin p to node i
g_i^p	the traffic flow arrived at i from origin p
c_r	the cost of path r, which is equal to total costs of all links on the paths
A_p	the restricting subnetwork for origin p
$B(i)$	the set of all directed links whose head node is i
b_i^p	the basic link of the links in $B(i)$ for origin p
$o^p(i)$	the topological order of node i for A_p
lcn_i^p	the last common node of all paths from origin p to node i
R_{ij}	the set of paths connecting i and j
$R_{ij}[A_p]$	the set of paths in A_p connecting i and j
f_{ij}^r	the flow on path r connecting i and j
$\delta_{ij}^{r,a}$	the path-link incidence matrix
q_{ij}	the total traffic demand between i and j, when $i = j$, $q_{ij} = 0$.

2.1 User Equilibrium Formulation

It is well known that TAP can be formulated as an optimization program with a nonlinear objective function and linear constraints. Consider an urban transportation network represented by a directed graph $G = (\mathcal{N}, \mathcal{A})$. For convenience, we suppose $O = D = \mathcal{N}$. Using the above notes, the User Equilibrium (UE) traffic assignment problem can be written as

$$\begin{aligned} \min\ & Z = \sum_{a\in\mathcal{A}} \int_0^{x_a(\alpha)} c_a(\varpi)d\varpi & \\ s.t.\ & \sum_{a\in B(i)} \alpha_a^p = 1, & 1 \le i, p \le N, i \ne p \\ & \alpha_a^p \ge 0, & \forall a \in \mathcal{A}, 1 \le p \le N \end{aligned} \tag{1}$$

where $\alpha = (\alpha^1, \alpha^2, \cdots, \alpha^N)$, $\alpha^p = (\cdots, \alpha_a^p, \cdots)$,$a \in \mathcal{A}$ is the variable of the mathematical programming, and α_a^p is defined as,

$$\alpha_a^p = \frac{x_a^p}{\sum_{b\in B(a_h)} x_b^p} \tag{2}$$

especially when the total traffic flow from origin p that arrived at a_h is zero, we let any one of the proportions of those links with head node a_h be one and the others be zero. $x_a(\alpha)$ is defined as,

$$x_a(\alpha) = \sum_{i=1}^{N} x_a^i(\alpha) = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{r\in R_{ij}} f_{ij}^r \cdot \delta_{ij}^{r,a} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{r\in R_{ij}} q_{ij} \prod_{b\subseteq r} \alpha_b^i \delta_{ij}^{r,a} \tag{3}$$

For each origin p and every node $j \ne p$ we choose one link $b_j^p \in A_p : (b_j^p)_h = j$ as the basic link of the links with head node j for origin p, call all other links with head node j in the restricting subnetwork A_p(if there are any) the non-basic links for origin p and denote them as $\mathtt{NB}_j^p = \{a \in A_p : a_h = j; a \neq b_j^p\}$; $\mathtt{NB}^p = \cup_{j \in \mathcal{N}; j \neq p} \mathtt{NB}_j^p$. Then α can be viewed as a function of $\alpha^{\mathtt{NB}}$:

$$\alpha_{b_j^p}^p(\alpha^{\mathtt{NB}}) = 1 - \sum_{a \in \mathtt{NB}_j^p} \alpha_a^{p\mathtt{NB}}, \quad \forall j \in \mathcal{N} \setminus \{p\}, \forall p \in \mathcal{N} \tag{4a}$$

$$\alpha_a^p(\alpha^{\mathtt{NB}}) = \alpha_a^{p\mathtt{NB}}, \qquad \forall a \in \mathtt{NB}^p, \forall p \in \mathcal{N} \tag{4b}$$

$$\alpha_a^p(\alpha^{\mathtt{NB}}) = \alpha_a^{p\mathtt{NB}} = 0, \qquad \forall a \in \mathcal{A} \backslash A_p, \forall p \in \mathcal{N} \tag{4c}$$

Using (4), (1) can be simply changed into,

$$\begin{aligned} \min \ Z = \sum_{a \in \mathcal{A}} \int_0^{x_a(\alpha^{\mathtt{NB}})} c_a(\varpi) d\varpi \\ s.t. \ \ \alpha_a^{p\mathtt{NB}} \geq 0, \qquad \forall a \in \mathcal{A}, 1 \leq p \leq N \end{aligned} \tag{5}$$

It is simply to prove that (1) equals to the famous Beckmann's transformation [2]. Convergence algorithms for solving it have been studied since the 1960s.

2.2 Origin-Based Algorithm

The origin-based algorithm operates directly on the space of the traffic flow proportion. It makes successive moves as an approximate Newton direction at each iteration. The main body of the algorithm can be found in [14].

3 Modified Origin-Based Algorithm

Our plentiful computational experiments and investigation of OBA indicate that there is no efficient and quick methods to find the last common nodes of the restricting subnetwork $A_p(\alpha)$ and we cannot but enumerate all paths of the restricting subnetwork in the algorithm. We also found the last common nodes were only used to calculate the approximation of the Hessian matrix. Hence if we can propose a new Hessian matrix approximation without using the last common nodes to replace that of the OBA, we will avoid path enumeration. Below the lower bound and upper bound of the Hessian matrix will be calculated firstly, then a novel Hessian matrix approximation is determined by these bounds. The novel approximation will be used to modify the origin-based algorithm.

Before giving the bounds of the Hessian matrix, we need the following definitions and propositions.

Definition 1. *For any origin p, and any two nodes i and j, we define the following formal expression as the proportion of the flow from origin p that arrives at node j through node i.*

$$\chi_{i \to j}^p = \begin{cases} \sum_{r \in R_{ij}[A_p]} \left(\prod_{a \subseteq r} \alpha_a^p \right), & \forall p \in \mathcal{N}, i \neq j \\ 1, & i = j \end{cases} \tag{6}$$

Definition 2. *For any origin p and node j, the following equation is defined as the average cost from origin p to node j.*

$$\sigma_j^p = \sum_{r\in R_{pj}[A_p]:a\subseteq r} c_r \cdot \prod_{a\subseteq r} \alpha_a^p \tag{7}$$

Definition 3. *For any origin p and link a, the following equation is defined as and the average cost of link a for origin p.*

$$\mu_a^p = \sum_{r\in R_{pa_h}[A_p]:\ a\subseteq r} c_r \cdot \prod_{a'\subseteq r:\ a'\neq a} \alpha_{a'}^p = c_a + \sigma_{a_t}^p \tag{8}$$

Definition 4. *For any origin p, the topological order of all nodes in A_p is defined as a one-to-one function $o : \mathcal{N} \to \{1, 2, \cdots, N\}$ such that, $\forall a = [i, j] \in A_p \Rightarrow o^p(i) < o^p(j)$, and particularly $o^p(p) = 1$.*

Definition 5. *The total flow arrived at node j from origin p is defined as,*

$$g_j^p = \sum_{i=1}^{N} \sum_{\substack{r\in R_{pi}[A_p];\\ j\in r}} q_{pi} \prod_{a\subseteq r} \alpha_a^p = \sum_{i=1}^{N} q_{pi}\cdot\chi_{p\to j}^p\cdot\chi_{j\to i}^p = \sum_{i=1}^{N} q_{pi}\cdot\chi_{j\to i}^p,\quad \forall p\in\mathcal{N} \tag{9}$$

Using these definitions, the following equations can be easily gotten,

$$\sigma_j^p = \sum_{a\in A_p:\ a_h=j} \alpha_a^p \cdot \sum_{r\in R_{pj}[A_p]:\ a\subseteq r} c_r \cdot \prod_{a'\subseteq r:\ a'\neq a} \alpha_{a'}^p = \sum_{a\in A_p:\ a_h=j} \alpha_a^p \cdot \mu_a^p \tag{10}$$

$$\begin{aligned}\mu_a^p &= c_a + \sum_{r\in R_{pa_t}[A_p]:\ a\subseteq r} c_r \cdot \prod_{a'\subseteq r} \alpha_{a'}^p \\ &= c_a + \sum_{r\in R_{pa_t}[A_p]:\ a\subseteq r} \prod_{a'\subseteq r} \alpha_{a'}^p \cdot \sum_{e\subseteq r} c_e \\ &= c_a + \sum_{a'\in A_p} c_{a'} \cdot \chi_{p\to a'_t}^p \cdot \alpha_{a'}^p \cdot \chi_{a'_h\to a_t}^p \\ &= c_a + \sum_{a'\in A_p; o^p(a'_h)<o^p(a_h)} c_{a'} \cdot \alpha_{a'}^p \cdot \chi_{a'_h\to a_t}^p \end{aligned} \tag{11}$$

$$\begin{aligned}\frac{\partial x_{a'}}{\partial \alpha_a^p} &= \sum_{i=1}^{N} \sum_{r\in R_{pi}[A_p];a'\subseteq r;a\subseteq r} q_{pi} \cdot \prod_{e\subseteq r;e\neq a} \alpha_e^p \\ &= \begin{cases} \sum_{i=1}^{N} q_{pi} \sum_{r\in R_{pi}[A_p];a\subseteq r;a'=a} \prod_{e\subseteq r;e\neq a} \alpha_e^p = g_{a_h}^p, & a' = a \\ 0, & a'_h = a_h, a' \neq a \\ \alpha_{a'}^p \cdot \chi_{a'_h\to a_t}^p \cdot g_{a_h}^p, & o^p(a'_h) < o^p(a_h) \\ \chi_{a_h\to a'_t}^p \cdot \alpha_{a'}^p \cdot g_{a'_h}^p, & o^p(a'_h) > o^p(a_h) \end{cases}\end{aligned} \tag{12}$$

$$\frac{\partial x_{a'}}{\partial \alpha_a^{p\mathtt{NB}}} = \begin{cases} g_{a_h}^p, & a' = a \\ -g_{a_h}^p & a' = b_{a_h}^p \\ 0, & a'_h = a_h, a' \neq a, a' \neq b_{a_h}^p \\ \alpha_{a'}^p \cdot g_{a_h}^p \cdot (\chi_{a'_h \to a_t}^p - \chi_{a'_h \to b_{a_h}^p{}_t}^p), & o^p(a'_h) < o^p(a_h) \\ 0, & o^p(a'_h) > o^p(a_h) \end{cases} \tag{13}$$

Hence the first-order and second-order derivatives of the variables $\alpha^{\mathtt{NB}}$ can be describe as,

$$\begin{aligned} \frac{\partial Z}{\partial \alpha_a^{p\mathtt{NB}}} &= \sum_{a' \in \mathcal{A}} \frac{\partial Z}{\partial x_{a'}} \cdot \frac{\partial x_{a'}}{\partial \alpha_a^{p\mathtt{NB}}} = \sum_{a' \in \mathcal{A}} c_{a'} \cdot \frac{\partial x_{a'}}{\partial \alpha_a^{p\mathtt{NB}}} \\ &= c_a \cdot g_{a_h}^p - c_{b_{a_h}^p} \cdot g_{a_h}^p + \sum_{\substack{a' \in A_p; \\ o^p(a'_h) < o^p(a_h)}} c_{a'} \cdot \alpha_{a'}^p \cdot g_{a_h}^p \cdot (\chi_{a'_h \to a_t}^p - \chi_{a'_h \to b_{a_h}^p{}_t}^p) \\ &= g_{a_h}^p (\mu_a^p - \mu_{b_{a_h}^p}^p) \end{aligned} \tag{14}$$

$$\begin{aligned} \frac{\partial^2 Z}{\partial \alpha_a^{p\mathtt{NB}2}} &= \sum_{a' \in \mathcal{A}} \Big[\frac{\partial^2 Z}{\partial x_{a'}{}^2} \cdot \Big(\frac{\partial x_{a'}}{\partial \alpha_a^{p\mathtt{NB}}} \Big)^2 + \frac{\partial Z}{\partial x_{a'}} \cdot \Big(\frac{\partial^2 x_{a'}}{\partial \alpha_a^{p\mathtt{NB}2}} \Big) \Big] = \sum_{a' \in \mathcal{A}} c'_{a'} \cdot \Big(\frac{\partial x_{a'}}{\partial \alpha_a^{p\mathtt{NB}}} \Big)^2 \\ &= (c'_a + c'_{b_{a_h}^p}) \cdot g_{a_h}^p{}^2 + \sum_{\substack{a' \in A_p; \\ o^p(a'_h) < o^p(a_h)}} c'_{a'} \cdot \alpha_{a'}^p{}^2 \cdot g_{a_h}^p{}^2 \cdot (\chi_{a'_h \to a_t}^p - \chi_{a'_h \to b_{a_h}^p{}_t}^p)^2 \end{aligned} \tag{15}$$

Notice that,

$$\chi_{p \to i}^p \cdot \chi_{i \to j}^p = \sum_{\substack{r \in R_{pj}[A_p]; \\ i \in r}} \prod_{a \subseteq r} \alpha_a^p \leq \sum_{r \in R_{pj}[A_p]} \prod_{a \subseteq r} = \chi_{p \to j}^p$$

and for any node j and any origin p, it is easy to prove $\chi_{p \to j}^p = 1$ by Lemma 4 of [14], then,

$$0 \leq \chi_{i \to j}^p \leq 1, \quad \forall p \in \mathcal{N}, 1 \leq i, j, \leq N$$

Hence the upper bound of the Hessian matrix will be,

$$\frac{\partial^2 Z}{\partial \alpha_a^{p\mathtt{NB}2}} \leq c'_a \cdot g_{a_h}^p{}^2 + c'_{b_{a_h}^p} \cdot g_{a_h}^p{}^2 + \sum_{\substack{a' \in A_p; \\ o^p(a'_h) < o^p(a_h)}} c'_{a'} \cdot \alpha_{a'}^p{}^2 \cdot g_{a_h}^p{}^2 \tag{16}$$

Because,

$$\begin{aligned} \frac{\partial^2 Z}{\partial \alpha_a^{p\mathtt{NB}2}} &= c'_a \cdot g_{a_h}^p{}^2 + c'_{b_{a_h}^p} \cdot g_{a_h}^p{}^2 \\ &\quad + \sum_{a' \in A_p; o^p(a'_h) < o^p(a_h)} c'_{a'} \cdot \alpha_{a'}^p{}^2 \cdot g_{a_h}^p{}^2 \cdot (\chi_{a'_h \to a_t}^p - \chi_{a'_h \to b_{a_h}^p{}_t}^p)^2 \\ &\geq c'_a \cdot g_{a_h}^p{}^2 + \sum_{a' \in A_p; a' \in B(a_t)} c'_{a'} \cdot \alpha_{a'}^p{}^2 \cdot g_{a_h}^p{}^2 \cdot (1 - \chi_{a_t \to b_{a_h}^p{}_t}^p)^2 \\ &\quad + c'_{b_{a_h}^p} \cdot g_{a_h}^p{}^2 + \sum_{a' \in A_p; a' \in B(b_{a_h}^p{}_t)} c'_{a'} \cdot \alpha_{a'}^p{}^2 \cdot g_{a_h}^p{}^2 \cdot (\chi_{b_{a_h}^p{}_t \to a_t}^p - 1)^2 \end{aligned} \tag{17}$$

and there is no loop for any restricting network A_p, hence

$$\begin{aligned}\chi^p_{a_t \to b^p_{a_h\,t}} &= \alpha^p_{[a_t, b^p_{a_h\,t}]}\\ \chi^p_{b^p_{a_h\,t} \to a_t} &= \alpha^p_{[b^p_{a_h\,t}, a_t]}\end{aligned}$$

Therefore the lower bound of the Hessian matrix is

$$\begin{aligned}\frac{\partial^2 Z}{\partial {\alpha^{p\,\mathtt{NB}}_a}^2} \geq{}& c'_a \cdot {g^p_{a_h}}^2 + \sum_{\substack{a' \in A_p;\\ a' \in B(a_t)}} c'_{a'} \cdot {\alpha^p_{a'}}^2 \cdot {g^p_{a_h}}^2 \cdot (1 - \alpha^p_{[a_t, b^p_{a_h\,t}]})^2\\ &+ c'_{b^p_{a_h}} \cdot {g^p_{a_h}}^2 + \sum_{\substack{a' \in A_p;\\ a' \in B(b^p_{a_h\,t})}} c'_{a'} \cdot {\alpha^p_{a'}}^2 \cdot {g^p_{a_h}}^2 \cdot (\alpha^p_{[b^p_{a_h\,t}, a_t]} - 1)^2\\ \geq{}& c'_a \cdot {g^p_{a_h}}^2 + c'_{b^p_{a_h}} \cdot {g^p_{a_h}}^2 \end{aligned} \tag{18}$$

Using the bounds of the Hessian matrix, we can get the following approximation of Newton direction of the variables $\alpha^{\mathtt{NB}}$,

$$\varDelta d^n_a = \frac{\mu^n_{b^n_{a_t}} - \mu^n_a}{(c'_a + c'_{b^n_{a_t}}) \cdot g^n_{a_t}}, \forall a \in \mathtt{NB}^n, n \in D.$$

Thus for any origin p, we have,

$$\varDelta \alpha^{p\,\mathtt{NB}}_a = \begin{cases} \max\left\{ -\alpha^{p\,\mathtt{NB}}_a, \lambda \cdot \frac{\mu^p_{b^p_{a_h}} - \mu^p_a}{(c'_a + c'_{b^p_{a_h}}) \cdot g^p_j(\alpha^{\mathtt{NB}})} \right\}, & g^p_{a_h}(\alpha^{\mathtt{NB}}) > 0 \\ -\alpha^{p\,\mathtt{NB}}_a, & g^p_{a_h}(\alpha^{\mathtt{NB}}) = 0 \end{cases}, \quad \forall a \in \mathtt{NB}^p \tag{19a}$$

$$\varDelta \alpha^{p\,\mathtt{NB}}_a = 0, \qquad \forall a \in \mathcal{A} \backslash A_p \tag{19b}$$

where λ is a step size which will be determined by the following algorithm.

```
Modified Origin-based Algorithm
  Initialization
    For $p$ from 1 to $N$
      $A_p$ = tree of minimum cost paths from origin p
      $x^p$ = all-or-nothing assignment using $A_p$
      Calculate $\alpha^p$ by using (2)
    End for
  Main loop
    While $x$ does not satisfy the convergence condition do
      For $p$ from 1 to $N$
        Update restricting subnetwork $A_p$
        Update origin-based link flows for origin $p$
      End for
    End while
  Update restricting subnetwork for origin $p$
```

```
    Remove unused links from $A_p$
    Compute maximum cost $u_i$ from $p$ to $i$ for all $i \in \mathcal{N}$
    For $a = [a_t, a_h]$ in $\mathcal{A}$
      If $u_{a_t} < u_{a_h}$ add link a to $A_p$
    End for
    Find topological order for new $A_p$
    Update data structures
  **Update origin-based link flows for origin $p$**
    Using (14) and (18) to compute average costs and Hessian approximations
      For step size $\lambda = 2^{-k}$, $k = 0, 1, 2, 3, \cdots$
        Using (19) to compute flow shifts $\Delta\alpha^p$ for $\lambda$
        Projection and aggregate flow shifts
        If new value of objective function is less than the old then stop
      End for
    Apply flow shifts
    Update total link flows and link costs.
```

4 Numerical Examples

Below we will illustrate both OBA and MOBA by the following networks whose basic characteristics are presented in Table 1. All programs are coded in $C\sharp$ and executed on a microcomputer with P4 2.0G, 512M. For fair comparison, common data structures for storing network topology and links and same convergence criterion are used in both OBA and the modified algorithm(MOBA). The following steps are used for the comparison of algorithm performance.

Table 1. Basic Characteristics of Test Urban Networks

Netork	Origin	Destination	Node	Link	OD Pair
Sioux Falls	24	24	24	76	528
Barcelona	97	108	930	2,522	7,922

Step 1. Solve TAP by LCFW (a modified Frank Wlofe algorithm)[8], with very large number of iteration and highly strict convergence criterion. Tag its final solution as Ideal Optimal Solution(IOS) and denote it with $x^I = (\cdots, x_a^I, \cdots)$. The intuitive reason for choosing LCFW is that it outperforms the other algorithms(including FW, GP etc.[8]) and it can avoid path enumerations.

Step 2. Considering that there is only one flow pattern that minimizes program (1)[3], we use the following convergence criterion in both OBA and MOBA.

$$\varepsilon = \frac{\sum_{a\in\mathcal{A}}(x_a(n) - x_a^I)^2}{\sum_{a\in\mathcal{A}} {x_a^I}^2} \tag{20}$$

where $x_a(n)$ is the link flows of the n-th iteration of the algorithms. When ε is less than a given small positive constant such as $1.0 \times 10^{-n}, n = 0, 1, 2, \cdots$, the algorithms will stop.

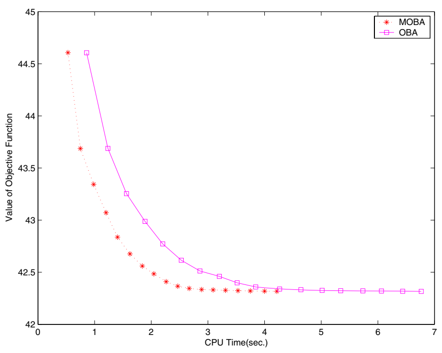


Fig. 1. OFV-CPU Time for Sioux Falls

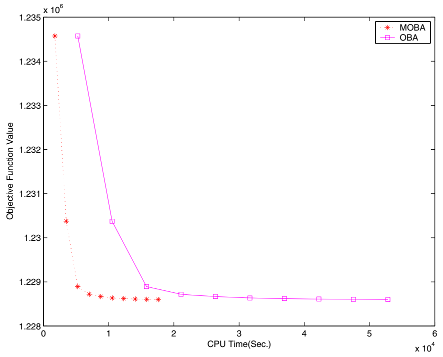


Fig. 2. OFV-CPU Time for Barselona

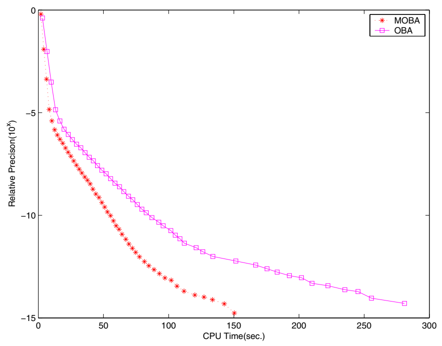


Fig. 3. $\log \varepsilon$-CPU Time for Sioux Falls

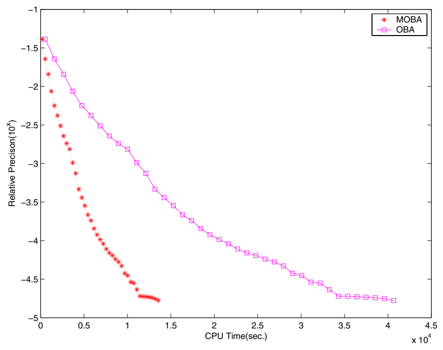


Fig. 4. $\log \varepsilon$-CPU Time for Barselona

Figure 1 and 2 show the relationship between the objective function value(OFV) and CPU time of the two algorithms for Sioux Falls network and Barcelona network. Figure 3 and 4 show the convergence of the two algorithms as measured by $\log \varepsilon$ vs. CPU time for those two real networks. From those four fights, we knew the performance of the modified origin based algorithm is better than that of origin-based algorithm. The modified algorithm is superior to the origin-basd algorithm in convergence and calculation time. At the same precision, the CPU time of the modified algorithm is only one third of the origin-based algorithm, even less than that. And the modified algorithm can solve the traffic assignment problem for large-scale network in an ideal time.

5 Conclusion

In this paper we provide a modififed origin-based algorithm based on esitimation of the lower and upper bounds of the Hessian matrix for, which can be calculated without path enumeration. In this algorithm we use the lower bound of Hessian matrix to approximate the direction of the origin-based algorithm. According to our computational results, the modified origin-based algorithm(MOBA) improves the convergence performance greatly. The results indicate that MOBA can deliver better and more reliable convergence than OBA and saves much more CPU time especially when large-scale networks are being considered.

References

1. Wardrop, J.G.: Some theoretical aspects of road traffic research. In: Proceedings of the institute of civil engineers, Part II, pp. 325–378 (1952)
2. Beckmann, M., Mcguire, C.B., Winster, C.B.: Studies in the economics of transportation. Yale University Press, New Heaven (1956)
3. Sheffi, Y.: Urban transportation networks: equilibrium analysis with mathematical programming methods. Prentice Hall, Inc., Englewood Cliffs (1984)
4. Leblanc, L.J., Morlok, E.K., Pierskalla, W.: An efficient approach to solving the road network equilibrium traffic assignment problem. Transportation Research 9, 309–318 (1975)
5. Leblanc, L.J., Helgason, R.V., Boyce, D.E.: Improved efficiency of the FrankWolfe algorithm for convex network problems. Transportation Science 19, 445–462 (1985)
6. Fukushima, M.: A modified frankwolfe algorithm for solving the traffic assignment problem. Transportation Research Part B 18(2), 169–177 (1984)
7. Anders, W., Carmen, O.: Accelerating convergence of the frankwolfe algorithm. Transportation Research Part B 19(2), 113–122 (1985)
8. Lee, D.H., Nie, Y.: Accelerating strategies and computational studies of the FrankWolfe algorithm for the traffic assignment problem. Transportation Research Record 1771, 97–105 (2001)
9. Gao, Z.Y., Lam, W.H.K., Wong, S.C., Yang, H.: The convergence of equilibrium algorithms with non-monotone line search technique. Applied Mathematics and Computation 148, 1–13 (2004)
10. Larsson, T., Patriksson, M.: Simplicial decomposition with disaggregated representation for the traffic assignment problem. Transportation Science 26(1), 4–17 (1992)
11. Bertsekas, D.: On the Goldstein-Levitin-Polyak gradient projection method. IEEE Transaction on Automatic Control 21, 174–183 (1976)
12. Jayakrishnan, R., Tsai, W.K., Prashker, J.N.: Faster path-based algorithm for traffic assignment. Transportation Research Record 1443, 75–83 (1994)
13. Lee, D.H., Nie, Y., Chen, A.: A conjugate gradient projection algorithm for the traffic assignment problem. Mathematical and computer modeling 37, 863–878 (2003)
14. Bar-Gera, H.: Origin-based algorithm for the traffic assignment problem. Transportation Science 36(4), 398–417 (2002)

Evaluation of Hierarchical Mesh Reorderings

Michelle Mills Strout[1], Nissa Osheim[1], Dave Rostron[1],
Paul D. Hovland[2], and Alex Pothen[3]

[1] Colorado State University, Fort Collins CO 80523, USA
[2] Argonne National Laboratory, Argonne IL 60439, USA
[3] Purdue University, West Lafayette IN 47907, USA

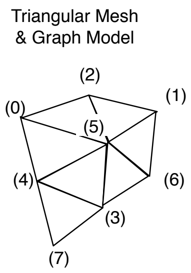
Abstract. Irregular and sparse scientific computing programs frequently experience performance losses due to inefficient use of the memory system in most machines. Previous work has shown that, for a graph model, performing a partitioning and then reordering within each partition improves performance. More recent work has shown that reordering heuristics based on a hypergraph model result in better reorderings than those based on a graph model. This paper studies the effects of hierarchical reordering strategies within the hypergraph model. In our experiments, the reorderings are applied to the nodes and elements of tetrahedral meshes, which are inputs to a mesh optimization application. We show that cache performance degrades over time with consecutive packing, but not with breadth-first ordering, and that hierarchical reorderings involving hypergraph partitioning followed by consecutive packing or breadth-first orderings in each partition improve overall execution time.

1 Introduction

Irregular scientific computing applications often achieve less than 10% of peak performance on current high performance computation systems, whereas on some systems dense matrix multiply can achieve more than 90% of peak performance. This gap in performance between dense (and regular) computations and sparse (and irregular) computations has been called the "sparse matrix gap." The sparse matrix gap can be attributed primarily not to poor scaling but to poor single-processor performance because of irregular memory references.

In this paper, we focus on irregular memory references due to indirect array addressing, which occurs in many applications such as partial differential equation solvers, molecular dynamics simulations, finite element analysis, mesh manipulation applications, and computations involving sparse matrix data structures. Figure 1 shows a loop with indirect array references that traverses triangular elements in a mesh. Each array entry in `data` could contain multiple fields such as the x and y coordinates of the corresponding vertex. Indirect memory references such as `data[n1[i]]` can exhibit poor data locality, and therefore cause performance problems. Improving data locality in irregular applications has been shown to improve parallel performance even more than serial performance [1,2,3,4].

```
// iterate over triangles
for (int i = 0; i < numTri; i++)
{
    // access data for
    // each vertex
    ... data[ n1[i] ] ...
    ... data[ n2[i] ] ...
    ... data[ n3[i] ] ...
}
```



Index and Data Arrays

	0	1	2	3	4	5	6
n1	(0)	(0)	(2)	(4)	(4)	(5)	(5)
n2	(5)	(2)	(1)	(3)	(5)	(1)	(6)
n3	(4)	(5)	(5)	(7)	(3)	(6)	(3)

	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
data	a	b	c	d	e	f	g	h

Fig. 1. Iteration over mesh elements, example triangular mesh, and index arrays storing the mesh. For each triangle i, the vertex indices are stored in $n1[i]$, $n2[i]$, $n3[i]$.

Previous research has studied various heuristic data and computation reorderings for improving data locality [2,5,6,7,8,9,10], but automatic determination of the reordering strategy that results in the best performance improvement remains an open problem. Toward solving this problem, this paper contributes a performance study of how hypergraph partitioning combined with a low overhead consecutive packing reordering [7] or a high performance hypergraph breadth-first ordering [10] affects the execution of a mesh optimization algorithm. In the context of Fig. 1, a data reordering involves reordering the entries in the `data` array. Each iteration of the loop in Fig. 1 is a computation. Figure 1 shows an example triangular mesh for use with the code in Fig. 1, where the data associated with one triangle is visited at each iteration of the loop. The data for each node in the mesh, (0), (1), etc., is stored in the corresponding entry in the *data* array. The *index* arrays store the mesh topology. An iteration, or computation, reordering involves rearranging the values in the index arrays and logically corresponds to permuting the order that triangles in the mesh are visited. Both the data and computation reordering problems can be modeled as the minimal linear arrangement problem, which is NP-complete.

Previous work on data reordering for irregular applications observed that *hierarchical*, or *hybrid*, heuristics can result in a 5 to 10% performance improvement over local heuristics alone [5]. Hierarchical heuristics perform a graph partitioning and then use a local reordering heuristic within each partitioning. The hierarchical technique proposed in [5] entails a graph partitioning , followed by a breadth-first ordering within each partition.

More recent research showed that local reordering heuristics (e.g., consecutive packing and breadth-first ordering) based on the hypergraph model perform up to 30% better than those based on a graph model in computations when three or more pieces of data are accessed within each iteration of the loop [10]. A hypergraph model groups any number of nodes into hyperedges. For the example in Fig. 1, each triangle could be represented with a hyperedge.

This paper studies the effect of hierarchical reordering in concert with a hypergraph-model based consecutive packing (Hyper-CPACK) or breadth-first ordering (Hyper-BFS) heuristics. We hypothesize that the performance of a computation when the data has been reordered using consecutive packing will degrade over the course of the computation. We also hypothesize that performing a hypergraph partitioning followed by a consecutive packing within each partition will improve performance because of the degradation within a partition having less effect than degradation over the full computation. For a breadth-first ordering based on a hypergraph model (Hyper-BFS), we hypothesize some improvement with hierarchical reordering, but not much because previous work [11] has shown that a breadth-first ordering using a hypergraph model achieves a high percentage of the memory bandwidth limit.

This paper makes the following contributions: algorithms for hierarchical reordering within the hypergraph model for both consecutive packing and breadth-first orderings within each partition; experimental results showing that cache performance degrades over time with consecutive packing and not with breadth-first on a hypergraph; and experimental results showing that hierarchical reordering on the hypergraph prevents the performance degradation since the local reorderings are performed within partitions.

2 Heuristics Using the Hypergraph Model

The distinguishing feature of hypergraphs are hyperedges, which are capable of connecting any number of nodes. Figure 2(a) shows the hypergraph that models relationships between the nodes in the mesh in Fig. 1. In the figure, the square vertices with parenthesized numbers directly correspond to nodes in the mesh. The filled-in squares are hyperedges connecting all the vertices of the element the hyperedge represents. Figure 2(b) is a dual hypergraph (i.e. each element in the mesh becomes a vertex and elements that share nodes are placed in the same hyperedge) for the hypergraph in Fig. 2(a). Hyper-BFS and the hierarchical version of Hyper-BFS use both the hypergraph and the dual hypergraph.

We use various combinations of six orderings for the experiments in this paper: original order, Hyper-BFS, Hyper-CPack, Hyper-Pack, HierBFS (Hierarchical BFS), and HierCPack (Hierarchical consecutive packing). We know from earlier work that heuristics based on hypergraph models outperform graph models [10]; we know hierarchical orderings are capable of limiting the growth of the working set, therefore in this work we hypothesize and show that combining hierarchical orders with hypergraph models do well.

Hypergraph Consecutive Packing (Hyper-CPack): One heuristic that most naturally generalizes to a hypergraph model is consecutive packing. The consecutive packing heuristic [7] packs the data associated with nodes in the hypergraph in the order that they are visited within the computation. For the example in Fig. 1, that means visiting the triangles in their given order and packing data for nodes as each node is seen in that order. It is critical that the tetrahedrons should be ordered well in this scheme, which is why our baseline mesh orderings start with tetrahedrons lexicographically sorted by the nodes they contain. Consecutive packing is commonly used because of its low overhead and reasonable resulting performance improvement.

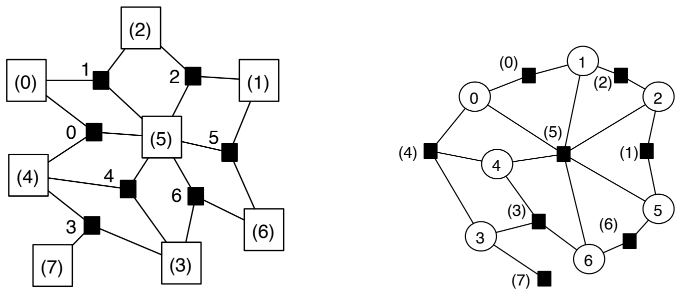


Fig. 2. The hypergraph (left) and dual hypergraph (right) models of the relationship between data and computation. The small black squares represent hyperedges. Numbers in parentheses represent nodes in the original mesh.

Hyper-CPack on the example in Figure 1 first orders the nodes of triangle 0: (0), (4), (5); then triangle 1: (2); and so on. The nodes in the first elements are guaranteed to be near each other, but the nodes in the later elements can be spread out because some of them have already been placed by earlier elements in which they are included. Therefore we hypothesize that the performance will deteriorate for the elements/iterations near the end of the ordering.

Hypergraph Breadth-First (Hyper-BFS): Hyper-BFS [10] also operates on the hypergraph. It starts at the first node and performs a breadth-first traversal, placing the nodes in the order the traversal visits them. Our experience suggests that the choice of starting node does not significantly affect the final performance. Hyper-BFS on the hypergraph visits all neighboring nodes that are part of the same hyperedge, before going on to other neighboring nodes. When Hyper-BFS orders a neighboring node, it orders all currently unordered nodes that are part of the same element before it orders other neighboring nodes.

Hypergraph Partitioning (Hyper-Part): Hypergraph partitioning decomposes the nodes of a hypergraph into disjoint sets. A reordering heuristic based on hypergraph partitioning then orders the nodes by partition. We have not done an extensive study, but differences between mesh partition quality given by various partitioners do not appear to have a significant effect our results. This is probably due to the fact that we are using the partitioners for single core data locality and not for parallelization. We use PaToH [12] as our hypergraph partitioner. If a hypergraph partitioner is used alone, the nodes and elements within each partition are left in their original order. If used as part of a hierarchical reordering, the local reordering is used within each partition created by a hypergraph partitioner. For these experiments, we set the size of the partitions so that the memory accesses of the computations for each partition fit into 1/2 of the L2 cache, following [13].

Hierarchical Consecutive Packing (HierCPACK): The hierarchical reordering heuristics, which are the new contributions of this paper, start with a hypergraph partitioning of the nodes and then perform a local reordering of the nodes within each partition. Hierarchical consecutive packing proceeds by visiting each hyperedge in the hypergraph model in the order that the hyperedges will be visited during run-time computations and packing the nodes in those hyperedges on a per partition basis. The algorithm visits all hyperedges in order and then maintains one packing lists of nodes for each partition. The final ordering concatenates all of the packing lists.

Hierarchical Breadth-First (HierBFS): Hierarchical Breadth-First also uses a hypergraph partitioning for hierarchical reordering. A breadth-first traversal over the nodes in each partition provides the local ordering. As with the non-hierarchical breadth-first ordering over a hypergraph, both the primal and dual hypergraphs are used to perform this reordering.

The algorithm first selects a root node for the breadth-first traversal from each partition. Next, it loops though the partitions, and for each partition it uses a queue data structure to perform a breadth-first traversal of the nodes based on adjacent hyperedges. Then, it loops through all unvisited neighbors in all the hyperedges and adds those to the new ordering and a queue. When it has searched through all the hyperedges for the root node, which can be found by accessing the dual hypergraph, it repeats the process for the next node in the queue. This process continues until all nodes in the partition have been added to the new ordering or it runs out of nodes in the queue. If the queue runs out before all the nodes in the partition have been visited, it searches the nodes for one in this partition that has not yet been visited and uses it as a new root node.

3 Experimental Results

We test the efficacy of the data and iteration/element reorderings by reordering real mesh data sets and feeding the reordered meshes into the FeasNewt mesh-quality optimization benchmark [11]. FeasNewt optimizes the quality of the tetrahedra by adjusting the coordinates of the internal mesh vertices. Higher-quality tetrahedra improve the accuracy and speed of computations or simulations using a discretization method. This approach does not change the topology of the mesh or the external shape.

FeasNewt has calculations and memory access patterns similar to those found in many scientific computing applications [11]. FeasNewt's gradient evaluation, Hessian computation, and a sparse matrix-vector product take the majority of its execution time. The gradient evaluation and Hessian computations iterate over mesh elements while the matrix-vector product operates on a sparse matrix with a row for each mesh node with non-zero blocks for higher-numbered neighboring mesh nodes. Although FeasNewt iteratively optimizes the mesh quality until convergence is reached, none of the reorderings used in this study alter the number of convergence iterations.

For input, we use six irregular tetrahedral meshes modeling different physical entities and from varying mesh generators (see Table 1). The sources of the

Table 1. Mesh data-set information

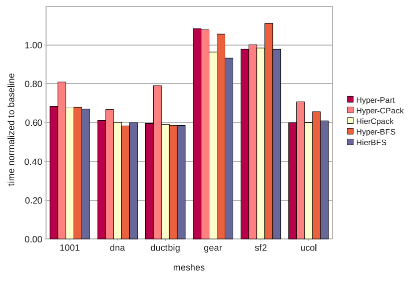
Mesh	# Nodes	# Elements	Size in MB	Comments
1-001.mesh	120,399	725,258	25.4	INRIA and TetGen
dna.mesh	185,823	938,168	33.3	INRIA and TetGen
ductbig.mesh	177,887	965,759	31.5	CUBIT
gear.mesh	285,640	1,595,392	58.3	CUBIT
sf2.mesh	378,747	2,067,739	61.6	CMU UMS
ucol.mesh	477,977	1,955,366	67.0	BioMesh

meshes include tetrahedral volume mesh generated using TetGen [14] using surface meshes from the INRIA Gamma team research database [15] (INRIA and TetGen), meshes generated using CUBIT [16], a mesh of the San Fernando Valley in Southern California from the CMU Unstructured Mesh Suite [17] (CMU UMS), and a mesh of parts of the circulatory system, generated as part of BioMesh Project [18], courtesy Chaman Singh Verma at Argonne. For the original ordering (baseline), the mesh node ordering provided by a mesh generator is used and all mesh elements are lexicographically sorted.

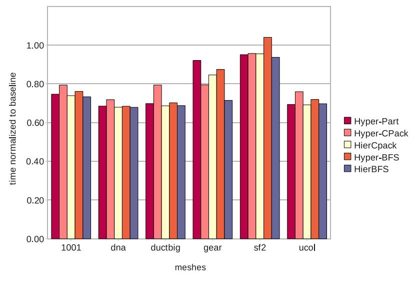
Our experiments were performed on a quiescent HP-xw9300 with 2 GB of memory and dual 64-bit AMD Opteron 250 2.4 GHz processors with 128 KB L1 cache and 1 MB L2 cache per processor. The code is single-threaded and therefore uses only one of the processors.

3.1 Effect of Hierarchical Reordering

Hierarchical data reordering improves performance over local reordering strategies alone (Hyper-CPack and Hyper-BFS), although for Hyper-BFS the improvement is minimal. Execution times for the full FeasNewt benchmark and the Hessian computation only are shown in Fig. 3(a) and 3(b), respectively. The execution times are normalized to the execution time for the benchmark when the original ordering is used and shown for each mesh and reordering strategy.



(a) FeasNewt benchmark.



(b) Hessian computation only.

Fig. 3. Normalized execution times for various data reorderings on a number of input meshes. All data reorderings are followed by Hyper-CPack iteration reordering.

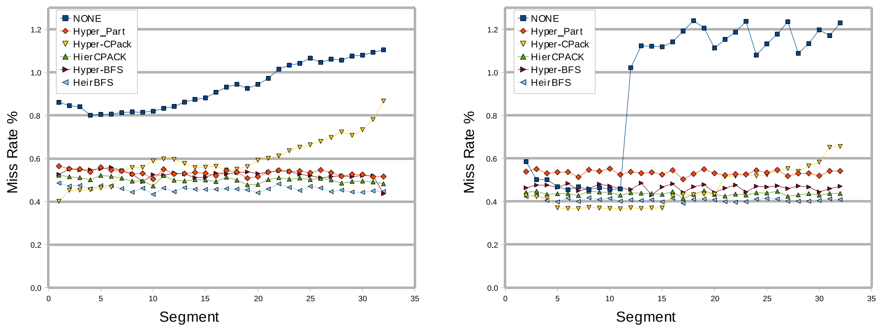


Fig. 4. L1 miss rates by segment over a run for ductbig (left) and dna (right). All data reorderings are followed by a Hyper-CPACK iteration reordering.

The hierarchical reordering strategies show the most improvement for most of the meshes for full FeasNewt benchmark execution times. In some cases, the performance improvement over the original mesh ordering is 40%.

One observation for Fig. 3(a) is that although the hierarchical reordering clearly improves over the local ordering consecutive packing, the hierarchical reorderings do not significantly improve over a global ordering based on hypergraph partitioning. One exception to this is the gear mesh, where hierarchical reorderings are the only reorderings that do not cause a slowdown. Possible future work is determining whether hierarchical reorderings can be "proven" safe from the standpoint of never causing slowdown.

3.2 Fine-Grained Cache Miss Results

We now demonstrate why hierarchical reordering improves over a consecutive packing alone. The performance due to a consecutive packing degrades over time, and hierarchical reordering evens out the performance benefits by doing consecutive packing within localized partitions. We observe this degradation by using a novel approach to studying the effect of data reordering. Specifically, we break the loop over tetrahedrons in the Hessian computation into equal-sized segments and record the cache miss rates for each segment. This approach exposes the data ordering quality as the computation progresses through the iteration ordering.

We use PAPI [19] to instrument FeasNewt to record L1 cache, L2 cache, and TLB hits and misses at 32 regular intervals, hereafter called segments, in the Hessian computation. The Hessian computation is performed multiple times per outer convergence loop. We therefore record a weighted running average for each segment's measurements. Execution times for these segments as well as the overall benchmark execution time were recorded. We present the minimum execution time from three runs.

To determine if Hyper-CPack and Hyper-BFS degrade over time and if hierarchical reordering with HierCPack and HierBFS stop this degradation, we observe the L1 cache miss rate by segment over the runs. Figure 4 shows L1 cache miss rates for the 32 segments of the Hessian loop. The graphs show the data reorderings for the ductbig and dna. The trends seen in ductbig and dna are typical of the trends seen in the other meshes, except for sf2. sf2 does not show much improvement in the cache miss rate over the segments, because it appears to already be well ordered.

Hyper-CPack does show the expected degradation over time, and HierCPack levels out this degradation and reduces the the overall cache misses as hypothesized (See the triangles pointing down and the triangles pointing up in Fig. 4). HierCPack can eliminate the performance degradation of Hyper-CPACK. Unlike Hyper-CPACK, Hyper-BFS does not show a degradation. Nonetheless, HierBFS still has consistently lower miss rates than Hyper-BFS and marginally better performance.

4 Related Work

Making decisions among all of the reordering heuristics is an open problem. Some work has done comparison among subsets [2,3,8,20]. However, since such comparisons might be relevant only to the specific benchmarks and datasets in the study, no clear winner exists. Typically, an ordering is selected due to results from earlier work on similar problems, or the desire to keep reordering costs low. Other work has used metrics to select among various reorderings without comparing execution time, with some success [10].

Many earlier studies on memory system performance of irregular codes have focused on reordering for data locality and other optimizations for sparse matrix-vector multiplication [3,9,21,22]. The mesh optimization benchmark [11] used in our experiments includes a symmetric, blocked sparse matrix multiply as well as iteration over a large tetrahedral mesh data structure. We observe that orderings based on a model of computation over the mesh data structure also improve the performance of the sparse matrix-vector multiply. Techniques specific to sparse matrices such as register tiling [9] might lead to even further performance benefits in the mesh optimization benchmark.

Our work differs from previous research in that the effect of data reordering on execution time and the memory hierarchy is explored at a finer granularity and in the context of multiple real datasets for a single benchmark. Previous work [23] has looked at the degradation of performance as the relationship between nodes in a molecular dynamics application changes. The granularity that we look at is smaller, since we focus on segments of one sweep over the mesh.

This paper studies the detailed differences between two local reordering heuristics and a hypergraph partitioning heuristic coupled with a local reordering heuristic. Al Furaih and Ranka [5] showed experimentally that hierarchical reorderings within a graph model improve performance, and later research has used some form of partitioning followed by a reordering within each partition [1,24]. This paper provides a similar basis for performing hierarchical reordering in reorderings based on hypergraph models. Selecting between hierarchical reorderings on a graph model versus a hypergraph model remains an open problem, but we hypothesize that, based on previous comparisons between the two models [10], hierarchical reordering on the hypergraph model will prevail.

5 Conclusion

Reordering the data and computation within irregular applications is important for improved data locality and performance. This paper presents new hierarchical heuristics based on a hypergraph model of the data reuse between computations. The new heuristics, hierarchical consecutive packing and hierarchical breadth-first, depend on a hypergraph partitioning followed by local reorderings within each partition. Our results show that hierarchical consecutive packing does improve performance in comparison with consecutive packing alone. More detailed experiments show that consecutive packing degrades in performance later in the ordering. When partitioning is done before the consecutive packing, each partition degrades separately and the overall degradation is not as severe. Hierarchical reordering does not improve significantly over a breadth-first ordering of the nodes in a hypergraph. Based on hardware counter results, we conclude that a breadth-first reordering on the hypergraph model does not result in the same degradation as consecutive packing and therefore does not benefit much from the grouping provided by hypergraph partitioning.

Acknowledgements

This work was supported by the CSCAPES Institute (DE-AC02-06CH11357 at Argonne) funded through DOE's SciDAC program. We thank Kevin Depue and Jinghua Fu for their programming efforts that contributed to this paper. We thank Gail Pieper for proofreading a draft of this paper.

References

1. Gropp, W.D., Kaushik, D.K., Keyes, D.E., Smith, B.F.: Performance modeling and tuning of an unstructured mesh CFD application. In: Proceedings of the ACM/IEEE Conference on Supercomputing (2000)
2. Han, H., Tseng, C.: A comparison of locality transformations for irregular codes. In: Dwarkadas, S. (ed.) LCR 2000. LNCS, vol. 1915, pp. 70–84. Springer, Heidelberg (2000)
3. Oliker, L., Li, X., Husbands, P., Biswas, R.: Effects of ordering strategies and programming paradigms on sparse matrix computations. SIAM Review 44(3), 373–393 (2002)
4. Martin, M.J., Singh, D.E., Tourino, J.: Exploiting locality in the run-time parallelization of irregular loops. In: International Conference on Parallel Processing (ICPP), August 18-21 (2002)
5. Al-Furaih, I., Ranka, S.: Memory hierarchy management for iterative graph structures. In: Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, March 30–April 3, 1998, pp. 298–302 (1998)

6. Mitchell, N., Carter, L., Ferrante, J.: Localizing non-affine array references. In: Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, pp. 192–202 (October 1999)
7. Ding, C., Kennedy, K.: Improving cache performance in dynamic applications through data and computation reorganization at run time. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 229–241 (May 1999)
8. Mellor-Crummey, J., Whalley, D., Kennedy, K.: Improving memory hierarchy performance for irregular applications using data and computation reorderings. International Journal of Parallel Programming 29(3), 217–247 (2001)
9. Vuduc, R., Demmel, J.W., Yelick, K.A., Kamil, S., Nishtala, R., Lee, B.: Performance optimizations and bounds for sparse matrix-vector multiply. In: Proceedings of the ACM/IEEE Conference on Supercomputing, pp. 1–35 (2002)
10. Strout, M.M., Hovland, P.D.: Metrics and models for reordering transformations. In: Proceedings of the The Second ACM SIGPLAN Workshop on Memory System Performance (MSP), pp. 23–34 (June 2004)
11. Munson, T.S., Hovland, P.D.: The FeasNewt benchmark. In: The IEEE International Symposium on Workload Characterization (IISWC 2005) (October 2005)
12. Catalyurek, U., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems 10(7), 673–693 (1999)
13. Pingali, V.K., McKee, S.A., Hseih, W.C., Carter, J.B.: Computation regrouping: restructuring programs for temporal data cache locality. In: Proceedings of the 16th International Conference on Supercomputing, pp. 252–261 (2002)
14. Si, H.: TetGen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, http://tetgen.berlios.de/
15. INRIA Gamma team research database, http://wwwc.inria.fr/gamma/gamma.php
16. CUBIT, Geometry and Mesh Generation Toolkit, http://cubit.sandia.gov/
17. O'Hallaron, D.R., Shewchuk, J.R.: CMU Unstructured Mesh Suite, http://www.cs.cmu.edu/~quake/meshsuite.html
18. BioMesh Project, an all-hex meshing strategy for bifurcation geometries, http://www.unix.mcs.anl.gov/~csverma/BioMesh/biomesh.html
19. London, K., Dongarra, J., Moore, S., Mucci, P., Seymour, K., Spencer, T.: End-user tools for application performance analysis using hardware counters. In: International Conference on Parallel and Distributed Computing Systems (August 2001)
20. Ou, C., Gunwani, M., Ranka, S.: Architecture-independent locality-improving transformations of computational graphs embedded in k-dimensions. In: Proceedings of the International Conference on Supercomputing (1995)
21. Taylor, V.E.: Sparse matrix computations: implications for cache designs. In: Proceedings of the ACM/IEEE Conference on Supercomputing, pp. 598–607 (1992)
22. Toledo, S.: Improving the memory-system performance of sparse-matrix vector multiplication. IBM Journal of Research and Development 41(6), 711–725 (1997)
23. Han, H., Rivera, G., Tseng, C.W.: Software support for improving locality in scientific codes. In: 8th Workshop on Compilers for Parallel Computers (CPC 2000), Aussois, France (January 2000)
24. Badawy, A.H.A., Aggarwal, A., Yeung, D., Tseng, C.W.: Evaluating the impact of memory system performance on software prefetching and locality optimizations. In: International Conference on Supercomputing, pp. 486–500 (2001)

Minkowski Functionals Study of Random Number Sequences

Xinyu Zhang[1], Seth Watts[2], Yaohang Li[3], and Daniel Tortorelli[2]

[1] College of Engineering, North Carolina A&T State University, Greensboro, NC 27411
xzhang@ncat.edu

[2] Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801
{watts2,dtortore}@uiuc.edu

[3] Department of Computer Science, North Carolina A&T State University, Greensboro, NC 27411
yaohang@ncat.edu

Abstract. Random number sequences are used in a wide range of applications such as simulation, sampling, numerical analysis, cryptography, and recreation. The quality of random number sequences is critical to the correctness of these applications. Many statistical tests have been developed to test various characteristics of random number generators such as randomness, independence, uniformity, etc. Most of them are based on testing on a single sequence. When multiple sequences are employed in an application, their potential correlations are also concerned. In this paper, we explore the techniques of using the Minkowski functionals and their extensions, the Minkowski valuations, to study the mathematical morphology of two dimensional binary image generated by pair-wise random number sequences, and apply this method to describe and compare the properties of several well-known pseudo- and quasi-random number generators.

Keywords: Minkowski functionals, random number, random number test, point pattern.

1 Introduction

Random number sequences are desired to display no describable deterministic patterns, but follow a certain statistical distribution. The quality of a random number generator is usually measured by efficiency, uniformity, independence, randomness, reproducibility, and aperiodicity. To test the quality of random number sequences, many statistical tests suites are available [1, 2]. Most of these tests are designed for testing a single sequence. However, in many applications such as parallel Monte Carlo, multiple sequences are involved and the potential correlations among these sequences may also affect the correctness of these applications. To ensure random behavior across multiple random number sequences, studies of statistical testing on interleaving sequences [3] as well as averaging sequences have been used in the past [4]. In this paper, we explore the testing of sequence correlations in two dimensions (2D).

The study of the pair-wise random number sequence correlation is the foundation of the analysis of correlations among multiple sequences. To identify the correlations between sequences, a tool to describe point distribution in 2D quantitatively is required. Recently, the Minkowski functionals have been used to quantify patterns found in galaxies, neuronal cells, and metal foams as well as random point patterns [5, 6, 7]. In 2D, the Minkowski functionals correspond to area V_0, perimeter V_1, and Euler characteristic V_2, i.e.:

$$V_0 = \int_P d^2A, V_1 = \frac{1}{4}\int_{\partial P} d^1S, V_2 = \frac{1}{2\pi}\int_{\partial P} \kappa d^1S, \tag{1}$$

where κ denotes the curvature of along the boundary ∂P. The Minkowski functionals distill the complexity of an image into a small number of descriptors. However, they cannot fully describe morphological properties such as heterogeneity, symmetry, and anisotropy of images. To overcome the shortcoming, they are extended to the Minkowski valuations, which are the higher order moments of the Minkowski functionals [8]. The first- and second-order moments are of particular interest. Correspondingly in 2D, there are three first-order moments, $\mathbf{V_0}$, $\mathbf{V_1}$, and $\mathbf{V_2}$, also referred to as the Minkowski vectors, given by:

$$\mathbf{V}_0 = \int_P \mathbf{x} d^2A, \mathbf{V}_1 = \frac{1}{4}\int_{\partial P} \mathbf{x} d^1S, \mathbf{V}_2 = \frac{1}{2\pi}\int_{\partial P} \kappa \mathbf{x} d^1S, \tag{2}$$

where $\mathbf{x}$ is the position vector. Moreover, the Minkowski vectors are usually normalized by their associated Minkowski functionals for a better geometric interpretation. These so-called centroids are defined as:

$$\mathbf{p}_i = \mathbf{V}_i / V_i \quad (i = 0,1,2 \textit{ if } V_i \neq 0), \tag{3}$$

where $\mathbf{p}_0$ is the center of mass, $\mathbf{p}_1$ is the center of perimeter, and $\mathbf{p}_2$ is the center of curvature. The second-order moments define the second-order Minkowski tensors, $\mathbf{V}_0^{2,0}$, $\mathbf{V}_1^{r,s}$, and $\mathbf{V}_2^{r,s}$, viz:

$$\mathbf{V}_0^{2,0} = \int_P \mathbf{x} \otimes \mathbf{x} d^2A, \mathbf{V}_1^{r,s} = \frac{1}{4}\int_{\partial P} \mathbf{x}^r \otimes \mathbf{n}^s d^1S, \mathbf{V}_2^{r,s} = \frac{1}{2\pi}\int_{\partial P} \kappa \mathbf{x}^r \otimes \mathbf{n}^s d^1S, \tag{4}$$

where $(r,s) = (2,0), (1,1), (0,2)$ indicates the degree of the tensor product of each vector with itself. However, only four of the above seven tensors carry independent information [9] and we will concentrate on the mass and perimeter tensors:

$$\text{mass tensor : } \mathbf{V}_0^{2,0} = \int_P \mathbf{x} \otimes \mathbf{x} d^2A, \tag{5}$$

$$\text{perimeter tensor : } \mathbf{V}_1^{2,0} = \frac{1}{4}\int_{\partial P} \mathbf{x} \otimes \mathbf{x} d^1S. \tag{6}$$

In this paper, we compare the Minkowski functionals of 2D binary images of the random number sequences generated by several well-known pseudo- and quasi-random number generators. We investigate the difference of the pseudo- and quasi-random number sequences from the theoretical values, and their Minkowski valuations. We also study the Minkowski functionals in 2D random point distribution when correlation between random number sequences occurs. The remainder of the paper is organized as follows: Sect. 2 describes the general method of Minkowski functionals study of random number sequences, Sect. 3 introduces the measures of the Minkowski functionals and valuations, Sect. 4 analyzes the sequences generated by the random generators mentioned above, and Sect. 5 finalizes the conclusions.

2 Computation of the Minkowski Functionals and Valuations

The definitions of Minkowski functionals are given in equation (1). For binary images on a square lattice, the Minkowski functionals are linear combinations of elements including faces, edges, and vertices. The computation of Minkowski functionals is simply counting the total number of faces n_2, edges n_1, and vertices n_0. The area, perimeter and Euler characteristic are computed by

$$V_0 = n_2, V_1 = -4n_2 + 2n_1,\ V_2 = n_2 - n_1 + n_0. \tag{7}$$

Michelsen *et al.* provided programming examples for counting n_2, n_1, and n_0 of 2D and 3D binary images [6]. Later Blasquez and Poraudeau gave a more efficient algorithm on 3D binary images by examining only half of a voxel's neighbors and using binary decision diagrams [10]. Their method can also be applied to 2D images.

In 2D, two random number sequences are mapped to *x* and *y* coordinates of the points in a square lattice of $L \times L$. The double precision random numbers are multiplied by *d* and truncated to get an integer in [0, *L*). Grains (discs or squares) are attached to the points and their sizes grow gradually. Figure 1 shows a square grain. As the lattices are square, it is reasonable to use square grains. A long sequence is divided into *k* subsequences and the Minkowski functionals are computed on each subsequence. As a result, this approach reveals both global and local properties.

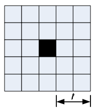


Fig. 1. Square grain with edge length of 2*r*+1

We consider a collection of *N* points whose *x* and *y* coordinates are generated from uniform, uncorrelated random number sequences. In the bulk limit, when the volume approaches infinity and the density ρ is fixed, the averages $\langle V_i / N \rangle_N$ are given by [7]:

$$\langle V_0 / N \rangle_N = (1 - e^{-\rho m_0}) / \rho, \tag{8}$$

$$\langle V_1 / N \rangle_N = m_1 e^{-\rho m_0}, \tag{9}$$

$$\left\langle V_2 / N \right\rangle_N = (m_2 - m_1^2 \rho) e^{-\rho m_0}, \tag{10}$$

where $\left\langle V_i \right\rangle_N$ denotes the average of the Minkowski functionals of the point ensemble with density ρ and m_i denotes the mean values of the Minkowski functionals of a single grain.

By applying the normalized Minkowski functionals on a square lattice, and substituting the Minkowski functionals for a single square grain of edge length *a*, the theoretical values of point distribution of uniform, uncorrelated random number sequences can be derived as

$$\left\langle A \right\rangle = 1 - e^{-n}, \left\langle U \right\rangle = 4a\rho^{1/2} e^{-n}, \left\langle \chi \right\rangle_N = (1-n)e^{-n}, \tag{11}$$

where $n = \rho a^2$ and A, U, χ, are the normalized Minkowski functionals ($A = V_0 / L^2, U = V_1 / LN^{1/2}, \chi = V_2 / N$) of a square [7]. Figure 2 shows the Minkowski functional curves with various densities. As the density decreases, the spans of the curves increase. While mapping a fixed length subsequence of random numbers to a lattice, a larger lattice size *L* yields a lower density and smoother curves, but demands more computation. In computation practice, one can control the density to be around 1% for a reasonable resolution.

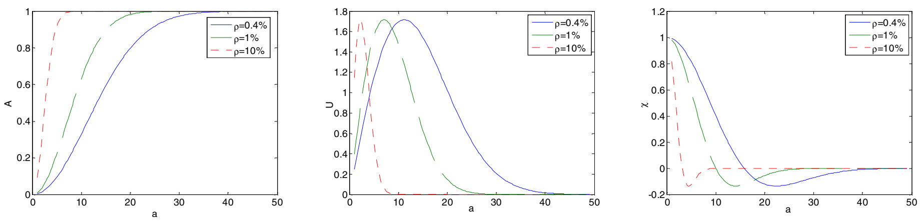


Fig. 2. Minkowski functionals as a function of square grain length

When computing the Minkowski valuations of 2D binary images, the equations (5) to (6) are used and the center of mass $\mathbf{p}_0$ and the mass tensor $\mathbf{V}_0^{2,0}$ are integrated on the pattern area, while the perimeter and curvature centroids $\mathbf{p}_1$ and $\mathbf{p}_2$, and perimeter tensor $\mathbf{V}_1^{2,0}$ are integrated on the pattern boundary [9]. An efficient algorithm is given by Zhang *et al.* [11].

3 Measures of the Minkowski Functionals and Valuations

As the grain size grows, the image will eventually cover the whole lattice. The grain size of the full coverage is related to the largest gap. The perimeter curve first increases and then decreases to zero due to most grains at smaller size are isolated and their growth mainly contributes to increasing perimeter. When the growth of grains reaches a certain size and the overlapping dominate the process, as a result, the perimeter curve starts to decrease and eventually drops to zero if periodic boundary conditions are applied. The Euler characteristic curve can be explained in a similar way. Initially, all of the grains are isolated and the Euler characteristic is 1. As the grains grow and overlapping occurs, the Euler characteristic decreases. When most of the grains overlap, the structure is dominated by holes and thus the Euler characteristic drops to negative. When the grains continue to grow, the holes start to be filled out and finally the Euler characteristic is 0 when the full coverage is reached.

Due to the discretization errors and the variations of random number generators, the experimental results may deviate from the theoretical values. To measure this difference, we compute the area between the two Minkowski functional curves using the trapezoid equation. Let $F(r)$ be the theoretical curve and $F_n(r)$ be the curve of a random number sequence, then the area is

$$D = \left(\sum_{i=1}^{r^*} \left| F(r_i) - F_n(r_i) \right| - \frac{\left(\left| F(r^*) - F_n(r^*) \right| + \left| F(r_1) - F_n(r_1) \right| \right)}{2} \right) \Delta r, \tag{12}$$

where r^* is the grain radius when the full coverage is reached for both curves.

In addition, we measure the distances dP of the centroids of the area, perimeter, or Euler characteristic **P** from the image center **C**,

$$dP = \left| \mathbf{P} - \mathbf{C} \right| \quad . \tag{13}$$

For a normalized square lattice with length 1.0, **C** is (0.5, 0.5).

Another important property of the random number sequences is the isotropy. The isotropy X is measured by the ratio of the two eigenvalues, λ_1 and λ_2, of a Minkowski tensor,

$$X = \left| \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \right| . \tag{14}$$

We measure the ratios of the eigenvalues of the area and perimeter tensors.

4 Random Number Generators

The middle-square method for random number generation was firstly suggested by John von Neumann. It has proved to be a comparatively poor source of pseudo-random numbers. The fundamental idea of the middle-square method is to take the square of the previous random integer and to extract the middle digits [2]. For example, one can take a four digit random integer 8653 and square it to get 74874409, and then take the middle 4 digits to create the next random integer of 8744. Most of the starting values will soon lead to a sequence with cycle of 6100, 2100, 4100, 8100, 6100 … or degenerate to zero. With more digits, the period is larger and the quality is better.

Linear Congruential Generator (LCG), 64 bit Linear Congruential Generator (LCG64), Multiplicative Lagged Fibonacci Generator (MLFG), Lagged Fibonacci Generator (LFG), and Combined Multiple Recursive Generator (CMRG) are well-known "good" pseudo-random number generators provided by the SPRNG (Scalable Parallel Random Number Generators) library [12]. Parameterization is used in SPRNG library to generate parallel, independent random number sequences.

Quasi-random number sequences, also called low-discrepancy sequences, are designed to improve convergence rate of Monte Carlo integration. The sequences intend to provide high uniformity instead of randomness to achieve low discrepancy. The Halton, Faure', and Sobol are popular quasi-random number generators [13].

In this article, we study the Minkowski functionals of the pseudo-random number sequences generated by LCG, LCG64, LFG, MLFG, and CMRG provided by the SPRNG library and the middle-square generator as well as the Halton, Faure', and Sobol quasi-random number sequences. For each of the pseudo- or quasi-random number generator, we generate 100 pairs of sequences with length 1,024,000. Each pair of these sequences is divided to 100 pairs of subsequences and mapped to the two axes of a square lattice of size 1024×1024 to form a 2D binary image. Square grains are attached to the points. The growth is cut off at r equals 100. Periodic boundary conditions are used for all the cases. The Minkowski functional measures given in the last section are computed with the results given in Tables 1, 2, and 3.

Table 1. Area bounded by the Minkowski functional curve from random number sequences and the theoretical uniform uncorrelated point distribution

	D_A		D_U		D_χ	
	Mean	Std	Mean	Std	Mean	Std
Pseudo-random number sequences						
Middle-square 6 digits	60.0637	29.1301	57.8705	35.7068	26.0772	29.9295
Middle-square 8 digits	80.3811	20.2021	45.9626	26.5424	41.1672	40.6423
LCG	0.0122	0.0071	0.4052	0.0062	0.2946	0.0088
LCG64	0.0122	0.0072	0.4055	0.0063	0.2947	0.0087
LFG	0.0123	0.0072	0.4054	0.0063	0.2947	0.0088
MLFG	0.0123	0.0072	0.4053	0.0063	0.2949	0.0088
CMRG	0.0124	0.0072	0.4053	0.0063	0.2946	0.0088
Quasi-random number sequences						
Sobol	0.4451	0.2704	1.7844	0.7816	0.8696	0.3007
Halton	0.5408	0.2506	2.2789	0.8433	1.1447	0.3475
Faure	1.1830	0.9434	3.6864	2.1378	1.5648	0.3856

The middle-square generators have extreme large D values, since degeneration occurs in many cases. The generators in SPRNG show very close means and standard deviations in Table 1 and demonstrate good match with theoretical values of the Minkowski functionals. Figure 3 shows the distribution of D_A, D_U, and D_χ of the LCG samples.

The values of quasi-random numbers vary dramatically, since their high uniformity and poor randomness by design. As a result, some obvious patterns can be observed in Fig. 4 for Sobol and Faure' sequences. The averaged Minkowski functionals as a function of radius are plotted in Fig. 5 for the LCG, Sobol, Halton, and Faure' sequences. When degeneration happens in the middle-square generator, the density

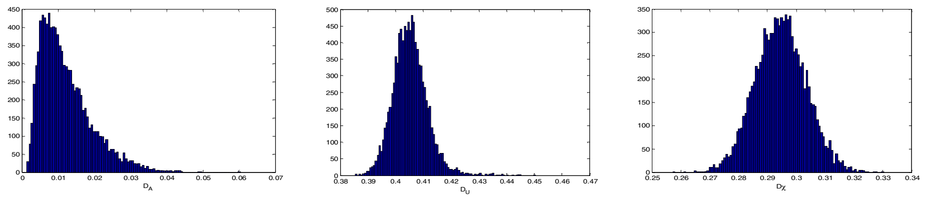


Fig. 3. Distributions of curve areas of the LCG sequences

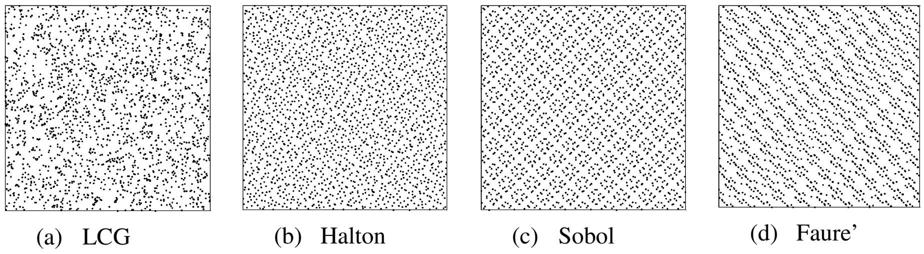
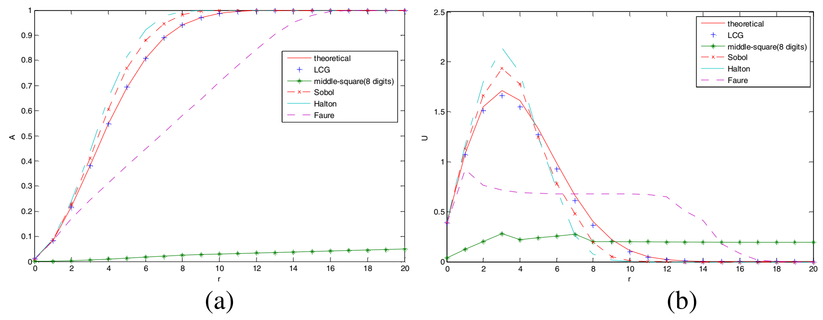


Fig. 4. 2D images of paired sequences generated from pseudo- and quasi-random number generators



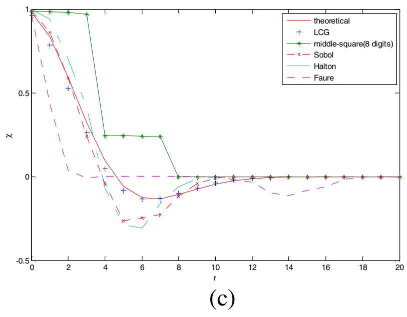


Fig. 5. Averaged Minkowski functionals as a function of the grain size

decreases and also causes a great deviation from the theoretical curve. The sequences such as Sobol and Halton with smaller gaps have their area values *A* grow faster and reach 1 (i.e., full coverage) at a smaller grain size, while the Faure' sequences have large gaps and reach the full coverage slowly. The degenerated patterns of middle-square reach their full coverage at a grain size greater than the cutoff radius 100.

The means and standard deviations of the distance of the centroids from the image center and the eigenvalue ratios are listed in Tables 2 and 3, respectively. The SPRNG pseudo-random number generators demonstrate similar characteristics. The quasi-random number generators yield rather different values due to their inherent patterns. The area and perimeter centroids of the quasi-random number generators are closer to the center with smaller variances, and the eigenvalue ratios of the quasi-random number sequences are smaller than pseudo-random numbers, which indicates better uniformity and isotropy.

Table 2. Distances of centroids from the image center

	dP_0		dP_1		dP_2	
	Mean	Std	Mean	Std	Mean	Std
Pseudo-random number sequences						
LCG	0.0013	0.0006	0.0907	0.0215	0.0894	0.0185
LCG64	0.0013	0.0006	0.0910	0.0223	0.0898	0.0191
LFG	0.0013	0.0006	0.0907	0.0221	0.0894	0.0189
MLFG	0.0013	0.0006	0.0910	0.0219	0.0898	0.0188
CMRG	0.0013	0.0006	0.0908	0.0221	0.0897	0.0189
Quasi-random number sequences						
Sobol	5.8681e-005	3.0018e-005	0.0589	0.0093	0.0622	0.0338
Halton	2.3473e-004	1.5721e-004	0.0753	0.0174	0.1336	0.3001
Faure	1.2252e-004	5.5186e-005	0.0465	0.0075	0.1911	0.3698

Table 3. Eigenvalue ratios of mass and perimeter tensors

	$X_{V_0^{2,0}}$		$X_{V_1^{2,0}}$	
	Mean	std	Mean	Std
Pseudo-random number sequences				
LCG	0.0018	0.0012	0.0835	0.0382
LCG64	0.0018	0.0012	0.0838	0.0383
LFG	0.0018	0.0012	0.0835	0.0388
MLFG	0.0018	0.0012	0.0837	0.0379
CMRG	0.0018	0.0012	0.0841	0.0388
Quasi-random number sequences				
Sobol	0.0002	0.0002	0.0096	0.0122
Halton	0.0004	0.0003	0.0340	0.0331
Faure	0.0001	0.0001	0.0055	0.0106

5 Correlation Identification

RANDU is an infamous pseudo-random number generator that has been used for decades on IBM mainframes [14]. If we simply generate random number doublets to build 2D binary images, it yields similar Minkowski values to those of the SPRNG generators provided in Tables 1-3. However, if we generate triplets *x,y,z* from a sequence and constructed a pair of random sequences in which one sequence comes from the *x*, and the other from *6x-9y+z*, all points fall into 15 lines in a stripe, as shown in Fig. 6(a), which indicates strong correlation [15]. In contrast, when a good random number generator is used, the points should fill in the stripe as shown in Fig. 6(b).

We compute the Minkowski functionals of the 2D binary images created by RANDU and LCG and CMRG in SPRNG, for 100 sequences and plot their averages as a function of the grain size. One can easily observe the difference between RANDU and SPRNG generators in Fig. 7. The area between the Minkowski functional curve and *x*-axis in Fig. 7(a) is 11.26 for RANDU, and 17.0359 for LCG and CMRG.

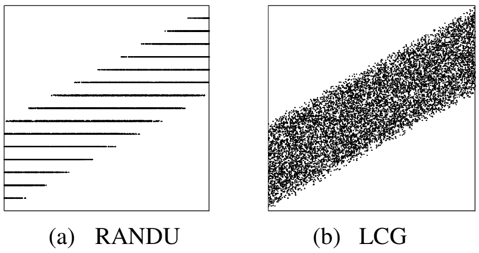


Fig. 6. Images of the RANDU and LCG sequences generated from *x* and 6*x*-9*y*+*z*

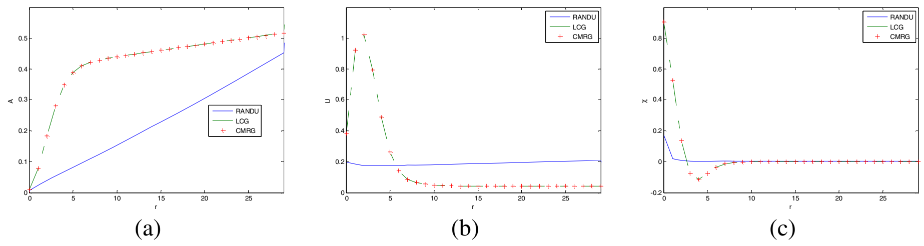


Fig. 7. Averaged Minkowski functionals as a function of grain size for RANDU, LCG, and CMRG sequences generated from *x* and 6*x*-9*y*+*z*

6 Conclusions and Discussions

In this paper, we discussed the method of applying the Minkowski functionals and valuations to study random number sequences. Pairs of sequences are mapped to 2D lattices to form a binary image. Grains are attached to the points. We compute the Minkowski functionals and valuations as a function of the grain size. A close match of the Minkowski functional curves with the theoretical curves of uniform uncorrelated point distribution indicates good randomness. The locations of the centroids of mass and perimeter, along with the ratios of mass and perimeter tensors give us some insight on the uniformity, symmetry, and isotropy of the patterns generated by random number sequences. Our examples given in this paper also showed that the Minkowski functionals are able to identify degenerated sequences, highly uniform sequences, and sequences with correlation. In summary, the Minkowski functionals and valuations can provide meaningful indication of the quality of random number sequences and are potential tools for testing new developed random number generators as a complementary to the existing statistical tests.

References

1. Soto, J.: Statistical Testing of Random Number Generators. In: Proceedings of the 22nd National Information Systems Security Conference (October 1999)
2. Knuth, D.: The Art of Computer Programming, vol. 2. Addison-Wesley, Reading (1969)
3. Cuccaro, S., Mascagni, M., Pryor, D.: Techniques for Testing the Quality of Parallel Pseudorandom Number Generators. In: Proceedings of the 7th SIAM Conf. on Parallel Processing for Scientific Computing, pp. 279–284. SIAM, Philadelphia (1995)
4. Coddington, P., Ko, S.: Techniques for Empirical Testing of Parallel Random Number generators. In: Proceedings of International Conference on Supercomputing, pp. 282–288 (1998)
5. Beisbart, C., Buchert, T., Wagner, H.: Morphology of spatial patterns. Physica A 293/3-4, 592–604 (2001)
6. Michielsen, K., Raedt, H., Hosson, J.: Aspects of Mathematical Morphology. Advances in imaging and electron physics 125, 119–194 (2002)
7. Michielsen, K., Raedt, H.: Integral-Geometry Morphological Image Analysis. Physics Reports 347, 461–538 (2001)
8. Beisbart, C.B.: Vector- and Tensor-Valued Descriptors for Spatial Patterns. In: Morphology of Condensed Matter, Physics and Geometry of Spatial Complex Systems, vol. 600(J), pp. 238–260. Springer, Berlin (2002)
9. Beisbart, C.: Measuring Cosmic Structure. Minkowski Valuations and Mark Correlations for Cosmological Morphometry, Dissertation, Ludwig-Maximi-lians-Universitat, Munchen (2001)
10. Blasquez, I., Poiraudeau, J.-F.: Efficient Processing of Minkowski Functionals on a 3D Binary Image using Binary Decision Diagrams. Journal of WSCG 11(1) (2003)
11. Zhang, X., Watts, S., Tortorelli, D.: Computation of First- and Second-order Minkowski Valuations for Two Dimensional Binary Images. Submitted to Applied Mathematical Modelling
12. Mascagni, M., Srinivasan, A.: Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation. ACM Transactions on Mathematical Software 26, 436–461 (2000)
13. Li, Y.: The Computational Measure of Uniformity, Master thesis, Florida State University (2000)
14. Zeitler, D., McKean, J., Kapenga, J.: Empirical Spectral Analysis of Random Number Generators. In: Proceedings of the 34th Symposium on the Interface. Montreal, Quebec, Canada (April 2002)
15. Marsaglia, G.: Random Numbers Fall Mainly in the Planes. PNAS 61, 25–28 (1968)

Autonomous Leaves Graph Applied to the Boundary Layer Problem

Sanderson L. Gonzaga de Oliveira and Mauricio Kischinhevsky

Instituto de Computação, Universidade Federal Fluminense,
Rua Passo da Pátria, 156, Bloco E, São Domingos, 24210-240, Niterói - RJ - Brazil
{sgonzaga,kisch}@ic.uff.br
http://www.ic.uff.br

Abstract. In physics and fluid mechanics, the boundary layer is the fluid layer in the immediate vicinity of a bounding surface. It is important in many aerodynamic problems. This work presents a numerical simulation of the bidimensional laminar boundary-layer problem considering a steady incompressible flow with no-slip condition on the surface by Autonomous Leaves Graph based on finite volume discretizations. In addition, a Modified Hilbert Curve numbers the control volumes. Initially, the numerical solution of the flat-plate problem is compared to its analytical solution, namely Blasius Solution. Secondly, simulations of the flow along a NACA airfoil shape are presented. Computer experiments show that an adaptive mesh refinement using the Autonomous Leaves Graph with the Modified Hilbert Curve numbering is appropriate for a aerodynamic problem. Finally, results illustrate that the method provides a good trade-off between speed and accuracy.

Keywords: Finite Volume Method, Adaptive mesh refinement, Boundary Layer Problem, NACA airfoils, Space-filling curves, Hilbert Curve.

1 Introduction

Numerical solution of partial differential equations (PDEs) may require the use of a mesh refinement strategy that concentrates more mesh points where the solution and/or its derivatives rapidly change. The Autonomous Leaves Graph (ALG) was proposed for the Finite Volume Method and a space-filling curve named Modified Hilbert Curve (MHC) was proposed in order to number the mesh control volumes [1]. ALG children nodes become autonomous as their parent node is deleted. Neighboring control-volume nodes which were generated from different parent volume nodes can be directly linked whether they have the same refinement level. On the other hand, they are indirectly linked through transition nodes in case they have different refinement levels. All modifications in the graph are merely local. In addition, an algorithm based on the Hilbert Curve construction is used in order to implement the numbering of the control-volume nodes.

Thus, this work presents a Finite Volume solution of the boundary layer problem using the ALG scheme. Boundary layers have been of great importance in the study of viscous fluid flow. In 1904, Ludwig Prandtl made the biggest breakthrough by demonstrating the existence of a thin boundary layer in fluid flow. Moreover, he found that there exists a thin layer near an object surface, where the viscous aerodynamic forces are as important as the inertial forces [2].

After this brief introduction, section 2 describes the Boundary Layer Problem. Next, section 3 shows the numerical method. Afterwards, section 4 shows the experimental results. Finally, section 5 draws some considerations.

2 Boundary Layer Problem

Let $\epsilon = \frac{\Delta}{L}$, where Δ is the velocity boundary-layer thickness and L is the reference length used in the Reynolds number [3]. Using a magnitude analysis order, no term in the y-momentum equation is larger than ϵ in the estimated magnitude and the well-known governing Navier-Stokes equations of viscous fluid flow can be greatly simplified within the boundary layer. Notably, the PDE characteristic becomes parabolic, rather than the elliptical form of the full Navier-Stokes equations. This greatly simplifies the solution of the equations. Thus, the Navier-Stokes equations for a bidimensional steady incompressible flow in Cartesian coordinates are given by the momentum and the continuity equations, i.e. the nonlinear governing PDEs in terms of dimensional variables are given by [3]

$$Continuity : \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0, \tag{1}$$

$$Momentum : u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{dp}{dx} + \nu\frac{\partial^2 u}{\partial y^2}, \tag{2}$$

where p is pressure, ρ is the fluid density, ν is the kinematic viscosity and u and v are the horizontal and vertical components of the vector field.

3 Description of a Discrete Formulation of the Boudary Layer Problem Based on the Finite Volume Method

This work considers the bidimensional control volume depicted in Fig. 1. Afterwards, the discretization of (1) and (2) are presented in the next sections.

3.1 Continuity Equation

In this work, the Finite Volume discretization of (1) is

$$Continuity : v_P^{k+1} - v_S^{k+1} = u_W^{k+1} - u_P^{k+1} \ . \tag{3}$$

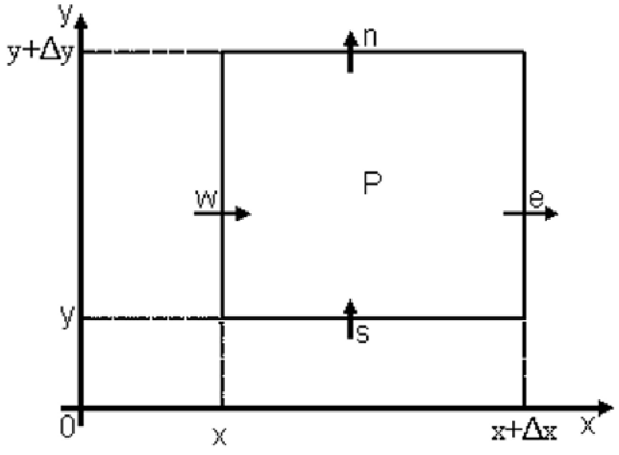


Fig. 1. Bidimensional control volume (adapted from [5])

3.2 Momentum Equation

The Finite Volume discretization of the momentum equation was divided in two parts: the discretization of the flat-plate Boundary Layer Problem and the Boundary Layer Problem with a NACA airfoil in the domain.

Discretization of the Flat-Plate Boundary Layer Problem. When it comes to the flat-plate Boundary Layer Problem, it has no pressure gradient flow [4]. Thus, since pressure is invariant, i.e. $\frac{dp}{dx} = 0$ because the inviscid flow over a flat plate yields a constant pressure over the surface, (2) can be rewritten as

$$Momentum : u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \nu\frac{\partial^2 u}{\partial y^2} \ . \qquad (4)$$

Integrating (4) in the control volume yields

$$Momentum : \int_w^e \int_s^n (u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y})dydx = \int_w^e \int_s^n \nu\frac{\partial^2 u}{\partial y^2}dydx \ . \qquad (5)$$

Applying the Divergence theorem, (5) yields

$$Momentum : \int_s^n u(u)\cdot \hat{n}_x dy + \int_w^e v(u)\cdot \hat{n}_y dx = \int_w^e \nu'\frac{\partial u}{\partial y}\cdot \hat{n}_y dx \qquad (6)$$

or

$$\int_s^n (uu|_e - uu|_w)dy + \int_w^e (vu|_n - vu|_s)dx = \int_w^e (\nu_n\frac{\partial u}{\partial y}|_n - \nu_s\frac{\partial u}{\partial y}|_s)dx \ . \qquad (7)$$

Assuming that the flow in the middle of the control volume edge represents the average of its variation in the edge [5], (7) can be rewritten as

$$\Delta y(uu|_e - uu|_w) + \Delta x(vu|_n - vu|_s) = \Delta x(\nu_n \frac{\partial u}{\partial y}|_n - \nu_s \frac{\partial u}{\partial y}|_s) \ . \tag{8}$$

Using an Upwind Differencing Scheme [6] yields

$$u_P u_P - u_P u_W + v_P u_P - v_P u_S = \frac{\nu}{h}(u_N - 2u_P + u_S), \tag{9}$$

where $h = \Delta x = \Delta y$ and $\nu = \nu_n = \nu_s$. Afterwards, (9) is divided by u_P for convenience. In addition, u_W is considered in the previous iteration. Thus, algebraic manipulations yield

$$u_P^{k+1} - u_N^{k+1}(\frac{\nu}{hu_P^k}) - u_S^{k+1}(\frac{v_P^k + \frac{\nu}{h}}{u_P^k}) = u_W^k - 2\frac{\nu}{u_P^k} - v_P^k, \tag{10}$$

where ν is the kinematic viscosity, h represents the vertical and horizontal edge size of a control volume. Including, (u_W, v_W), (u_N, v_N), and (u_S, v_S) are the west, north and south neighbors of the control volume (u_P, v_P), respectively.

To summarize, the here proposed Finite Volume equations are written in (3) and (10), which are numerical approximations of the mathematical modeling given in (1) and (4), respectively. Fig. 2 depicts the discretization scheme adopted where the circles represent the control volumes of the Finite Volume mesh.

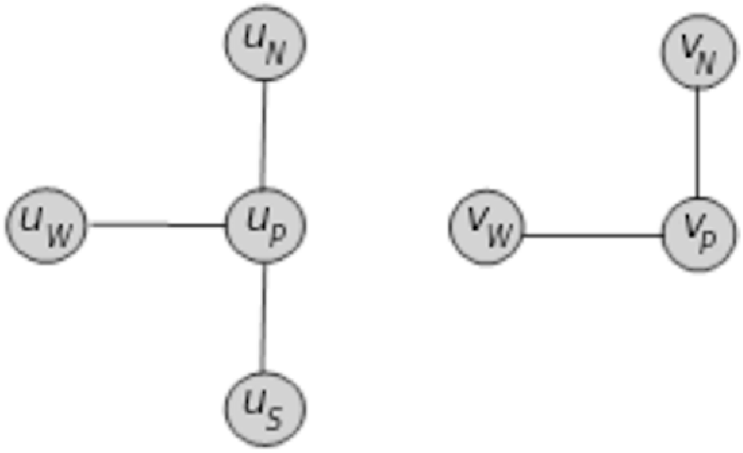


Fig. 2. Adopted discretization scheme of the momentum and continuity equations, respectively

Finite Volume Discretization of the Momentum Equation of the Boundary Layer Problem. The usual boundary conditions can be applied as where the subscript e in Fig. 3 refers to conditions on the Boundary Layer edge. The pressure gradient term in (2) is evaluated in the boundary-layer border. Since $u_e(x)$ is specified, $\frac{dp}{dx}$ can be evaluated applying equations that govern the inviscid outer flow (Euler's equation) [3]

$$\frac{dp}{dx} = -\rho u_e \frac{du_e}{dx} \ . \tag{11}$$

Taking this into account, the here proposed Finite Volume discretization of the momentum equation is written in (12), which is a numerical approximation of the mathematical modeling given in (2).

$$u_P^{k+1} - u_N^{k+1}\left(\frac{\nu}{hu_P^k}\right) - u_S^{k+1}\left(\frac{v_P^k + \frac{\nu}{h}}{u_P^k}\right) = u_W^k - 2\frac{\nu}{u_P^k} - v_P^k + u_{eP}\frac{u_{eP}^k - u_{eW}^k}{hu_P^k} \ . \tag{12}$$

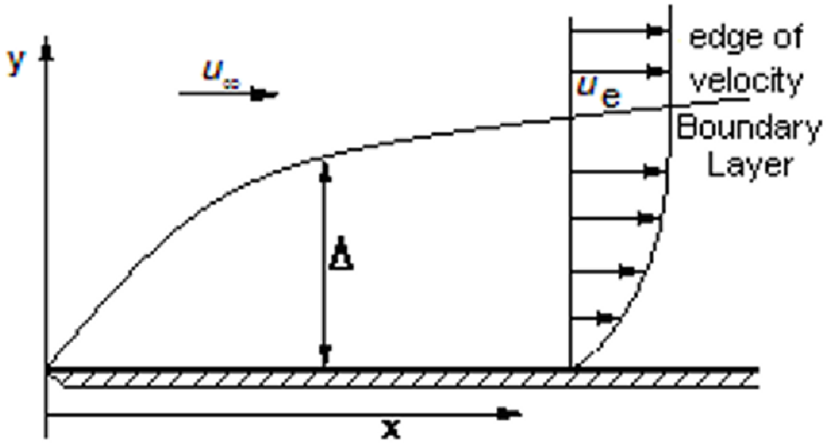


Fig. 3. Flat-plate Boundary Layer process (adapted from [3])

4 Experimental Investigation

Tests were performed with air kinematic viscosity $\nu = 1.5 \cdot 10^{-5}\frac{m^2}{s}$. Refinement decisions are performed in both 2D directions verifying the difference of the absolute value between two control volumes.

4.1 Flat-Plate Boundary Layer Numerical Simulation

The Analytical Solution. Since no pressure gradient exists and a constant Boundary Layer edge velocity occurs for this flow as well, all the profiles along the plate can be represented by a single curve with the proper dimensionless. For the solution of this constant property, the flat-plate flow is known as the Blasius Solution, which shows that for a flow with Reynolds number (Re) much larger than unity, i.e. Re$>>$1, the velocity profiles have the same dimensionless shape in the Boundary Layer region. More precisely, a dimensionless similarity variable in the normal direction is given by [7]

$$\eta = y\sqrt{\frac{u_\infty}{2\nu x}}, \tag{13}$$

where x, y are the distances from the flat-plate leading edge, u_∞ is the air velocity in the free-stream region, and ν is the air kinematic viscosity.

Subsequently, the parallel velocity to the plate is dimensionless by the edge velocity u_∞, which is the free-stream velocity for this particular case. Thus, the Boundary Layer thickness over a flat plate is given by [7]

$$\triangle = \frac{5x}{\sqrt{\frac{u_\infty x}{\nu}}} \ . \tag{14}$$

A Numerical Simulation. Tests of the flat-plate Boundary Layer Problem were performed with $u_\infty = 200$, $u_\infty = 10$ and a limit of eight refinement levels for each control volume. Initially, Blasius solutions are depicted in Fig. 4.

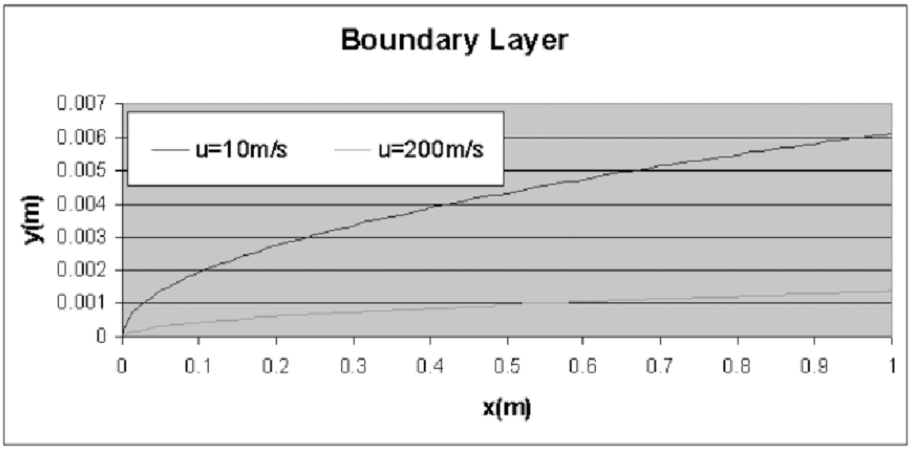


Fig. 4. Blasius Solution to $\nu = 1.5 \cdot 10^{-5} \frac{m^2}{s}$, $u_\infty = 200$ and $u_\infty = 10$

One example of the tests performed is depicted in Fig. 5 comprising of $x = [0; 1]$, $y = [0; 0.1]$ and $u_\infty = 200$. Fig. 5 shows the Modified Hilbert Curve for numbering the mesh control volumes. Besides, Fig. 5 shows the Boundary Layer in the bottom of the domain.

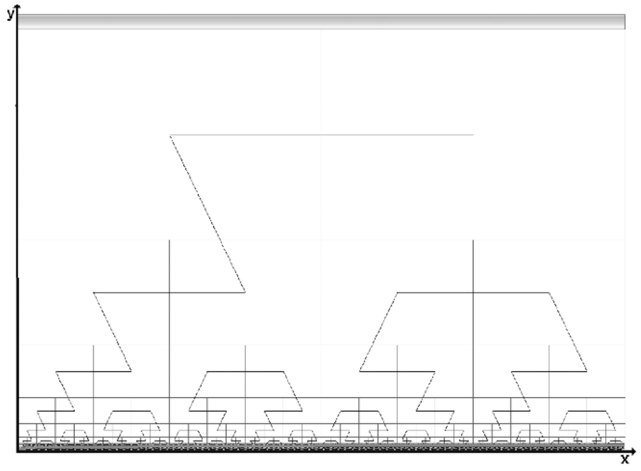


Fig. 5. Flat-plate Boundary Layer Problem with $\nu = 1.5 \cdot 10^{-5}$, $u_\infty = 200$ (x=[0;1], y=[0;0.1])

The numerical average error is the difference between the numerical simulation and the Blasius Solution

$$error = \frac{\sum_{i=1}^{n} |Simulation_i - Blasius_i|}{n}, \tag{15}$$

where n is the number of control volumes.

Tests comparing the adaptive mesh refinement (AMR) and the non-AMR solutions were performed. Exemplifying, the non-AMR used 340 control volumes whereas the test with the AMR, presented in Fig. 5, used 164 control volumes, which makes up 42.2% less control volumes in relation to the non-AMR test. Along the same lines, the AMR scheme spent 234 milliseconds whereas the non-AMR scheme spent 469 milliseconds. Considering the numerical average error between the simulation and Blasius solution, it is $5.62 \cdot 10^{-4}$ in such test.

Figure 6 depicts a test performed with x=[0;1], y=[0; 0.01] and $u_\infty = 10$ in order to show that the results correspond to the Blasius Solution. In this test, 340 control volumes were generated for the non-AMR scheme whereas the test performed to the AMR scheme generated 169 control volumes, which is 49.7% less of the total amount of control volumes in the non-AMR scheme. In addition, the AMR scheme spent 813 milliseconds whereas the non-AMR scheme spent 1282 milliseconds. In this test, a numerical average error between the numerical approximation and the Blasius Solution is $8.31 \cdot 10^{-3}$. Table 4.1 summarizes those results. Moreover, Table 1 presents a comparison of the number of control volumes and time between the adaptive and the non-adaptive schemes (processing time in milliseconds).

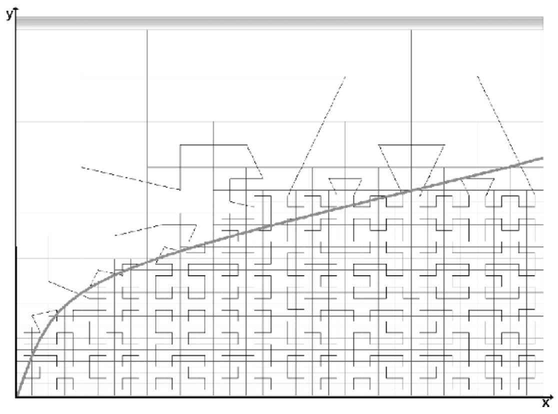


Fig. 6. Flat-plate Boundary Layer Problem with $\nu = 1.5 \cdot 10^{-5}$, $u_\infty = 10$ (x=[0;1], y=[0;0.01])

Table 1. Comparison between the adaptive and the non-adaptive schemes

u_∞	200		10	
Tests	Control volumes	Time	Control volumes	Time
Non-AMR	340	469	340	1282
AMR	164	234	169	813
Error	$5.62 \cdot 10^{-4}$		$8.31 \cdot 10^{-3}$	

4.2 Boundary Layer Numerical Simulation with a NACA Airfoil in the Domain

A NACA0012 airfoil is presented in the domain with 35^o of angle of attack. This test was performed with $u_\infty = 250$ and $u_\infty = 350$. Figure 7 shows a Boundary Layer numerical simulation with a NACA0012 in the domain and a limit of six refinement levels for each control volume. Figure 8 represents the vector field of the flux of a test. Figure 9 shows a test making up 7474 control volumes. This test was performed with $u_\infty = 350$ and a limit of 10 refinement levels for each control volume.

5 Consideration Remarks

This work presents a numerical simulation of the Boundary Layer Problem by ALG based on the Finite Volume Method. The flat-plate Boundary Layer numerical simulation is compared to the Blasius Solution. A NACA0012 airfoil is numerically simulated showing that ALG and MHC (this for numbering the control volumes) are adequate to simulate a complex problem such as in aerodynamics. In future works, ALG based on triangular control volumes should be investigated.

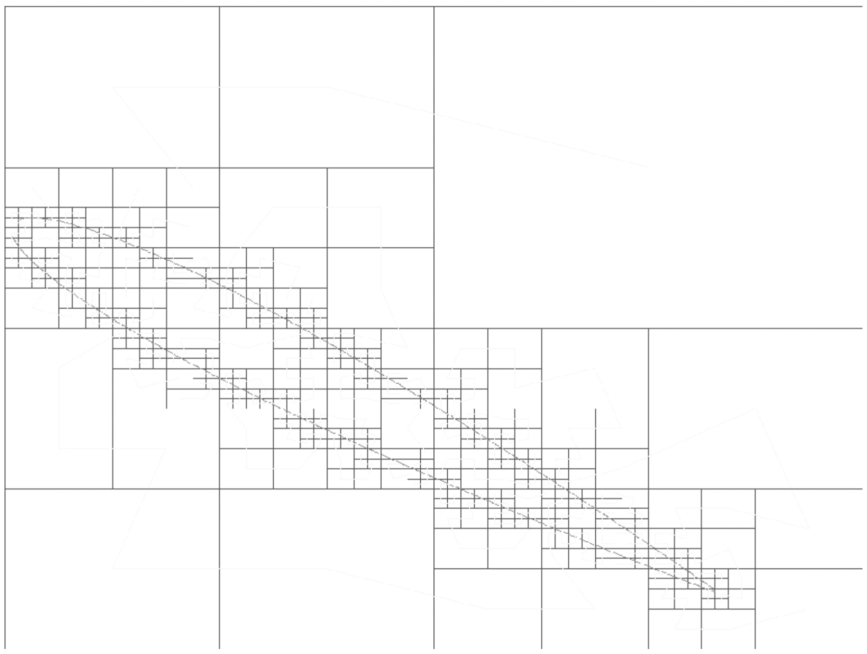


Fig. 7. Numerical simulation of the Boundary Layer Problem with a NACA0012 airfoil in the domain, $u_\infty = 250$ and 35^o of angle of attack (x=[0;1], y=[0;1])

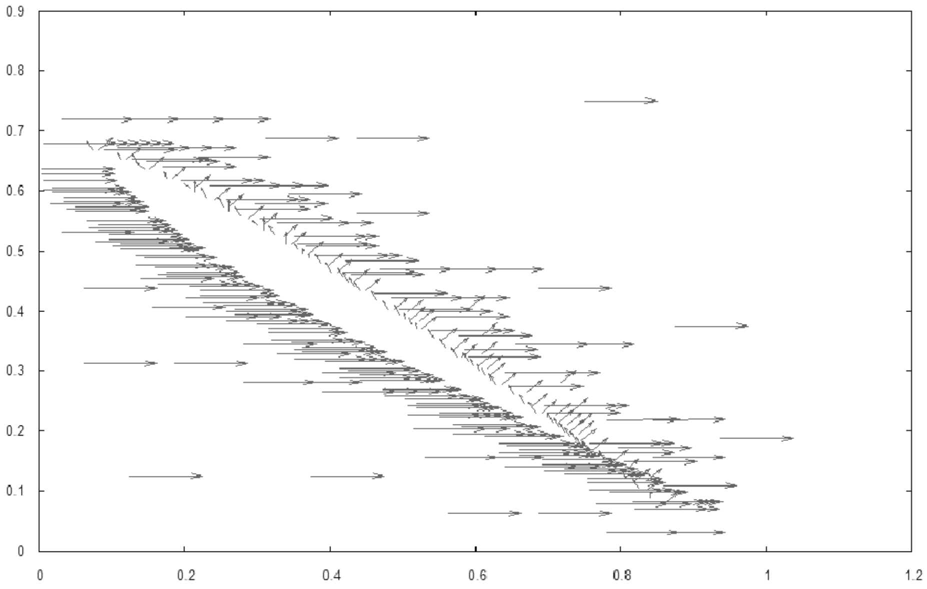


Fig. 8. Directions of the vector field (u, v)

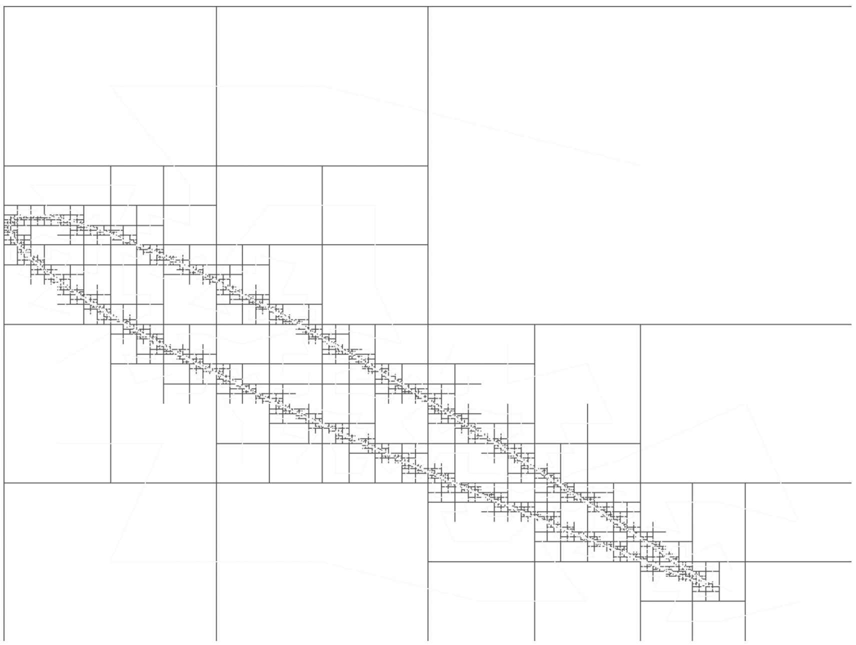


Fig. 9. Boundary Layer numerical simulation with the NACA0012 airfoil in the domain and $u_\infty = 350$

Acknowledgments. Authors gratefully acknowledge CAPES for the finantial support.

References

1. Burgarelli, D., Kischinhevsky, M., Biezuner, R.: A new adaptive mesh refinement strategy for numerically solving evolutionary PDE's. J. of Computational and Applied Mathematics. 196, 115–131 (2006)
2. Venkatachari, B.S.: Development and validation of a transient viscous flow solver based on a space-time CE/SE framework. The University of Alabama at Birmingham (2005)
3. Anderson, D.A., Tannehill, J.C., Pletcher, R.H.: Computational Fluid Mechanics and Heat Transfer. Hemisphere (1984)
4. Neel, R.E.: Advances in computational fluid dynamics: turbulent separated flows and transonic potential flows. Faculty of Virginia Polytechnic Institute and State University (1997)
5. Sperandio, D., Mendes, J.T., Silva, L.H.M.: Cálculo Numérico: Características Matemáticas e Computacionais dos Métodos Numéricos. Pearson Prentice Hall (2006)
6. Madsen, J.I.: Design optimization of internal flow devices. Alborg University (1998)
7. Schlichting, H.: Boundary-Layer Theory. McGraw-Hill Inc., New York (1979)

Finite-Element Non-conforming h-Adaptive Strategy Based on Autonomous Leaves Graph

Diego Brandão, Sanderson L. Gonzaga de Oliveira, and Mauricio Kischinhevsky

Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Pátria, 156, Bloco E, São Domingos, 24210-240, Niterói - RJ - Brazil
http://www.ic.uff.br

Abstract. Adaptive mesh refinement techniques are used in order to decrease the computational cost associated with the numerical solution of Partial Differential Equations. In this work, the refined mesh is represented by a graph data structure. More precisely. this scheme follows the Autonomous Leaves Graph concepts. The objective is to construct an adaptive mesh refinement with lower cost than tree-based schemes. Moreover, the Autonomous Leaves Graph was initially proposed with the Finite Volume Method and a Modified Hilbert Curve was used for the total-ordering of the control volumes. This work proposes to integrate the Autonomous Leaves Graph and the Finite Element Method as well as to adapt the Modified Hilbert Curve for this scheme. Furthermore, a non-conforming h-adaptive strategy is implemented. This approach is applied in the solution of the Poisson equation problem and the experimental results are discussed.

Keywords: Finite Element Method, h-adaptativity, Autonomous Leaves Graph, Space-filling curves, Numerical Methods, Hilbert Curve, Non-conforming meshes.

1 Introduction

Adaptive mesh refinement (AMR) strategies are important in several areas of engineering and uniform meshes have been widely used in order to solve such problems. However, uniform meshes are not a computationally viable choice for solving a system of Partial Differential Equation (PDEs) in which steep gradients, singularities, or discontinuities need to be captured. Thus, since a tree-based structure provides a simple scheme for h and p-refinement, it has been broadly used. In this structure, the parents and the refined children elements are represented in the same computational data structure, generating an overhead when scanning the neighbors elements in order to assemble the resulting linear system.

The research in this field had a relevant advance when the Autonomous Leaves Graph (ALG) was proposed for the Finite Volume Method and a space-filling curve, named Modified Hilbert Curve (MHC), was proposed in order to number the control volumes [1]. This present work proposes the ALG technique for the Finite Element Method (FEM) and an adaptation of the MHC for this scheme as well.

In the following section, the ALG is briefly presented. Next, Section 3 describes the MHC adaptation to FEM-ALG. Afterwards, Section 4 shows the numerical results. Finally, Section 5 draws some concluding remarks.

2 Data Structure

Consider a bidimensional domain in a unit square $\Omega = [0, 1]^2$. This square is discretized into four new equal ones, obtaining four square discrete places. The discretized domain is represented in Fig. 1.

The graph of Fig. 1 shows black circles, which are called here as black nodes. Black nodes represent the discrete places of the discretized domain in Fig 1. In

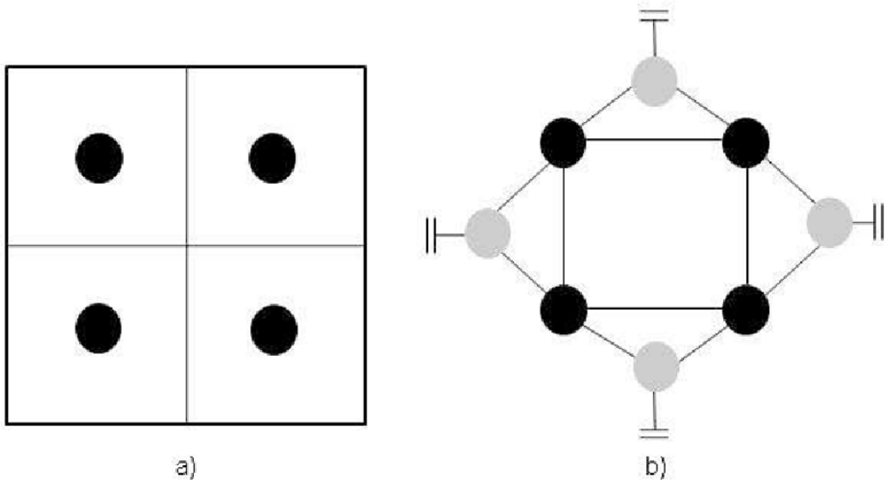


Fig. 1. An initial discretized domain and a graph that represents it

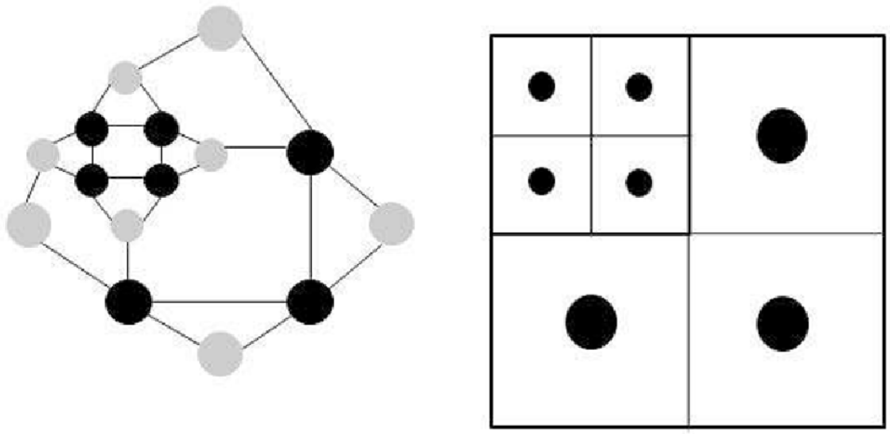


Fig. 2. Refinement process

addition, the gray circles of the graph represent transition nodes. They furnish connectivity information between nodes with different level of refinements. The ones in Fig. 1 represent connectivity information within the boundary of the domain.

In the refinement process, an element in a level n is replaced by four black nodes and four transition nodes in a level $n + 1$. This process is sketched in Fig. 2.

3 MHC Adaptation

The ALG was proposed as a graph data structure in which each node represents a control volume. Similarly, this present work proposes a graph data structure in which each node represents a finite element.

Since non-conforming meshes are not represented by the Hilbert Curve, the original ALG proposed the MHC in order to number the barycenter of the control volumes. The MHC was implemented by a linked list. Likewise, this present work implements two linked lists in order to number: *i)* the barycenter of the finite elements (likewise the original ALG)), i.e. a linked list to number the finite elements and *ii)* the vertices of the non-conforming finite elements. Moreover, since in the FEM the values are evaluated in the element vertices, the second linked list is an adaptation of the MHC algorithm in order to number the vertices of the finite elements. Figure 3 depicts an example of the vertice numbering by the adaptation of the MHC.

Each node of the second linked list represents a mesh vertex as well as retains relevant information about which finite element it belongs to. Furthermore, each

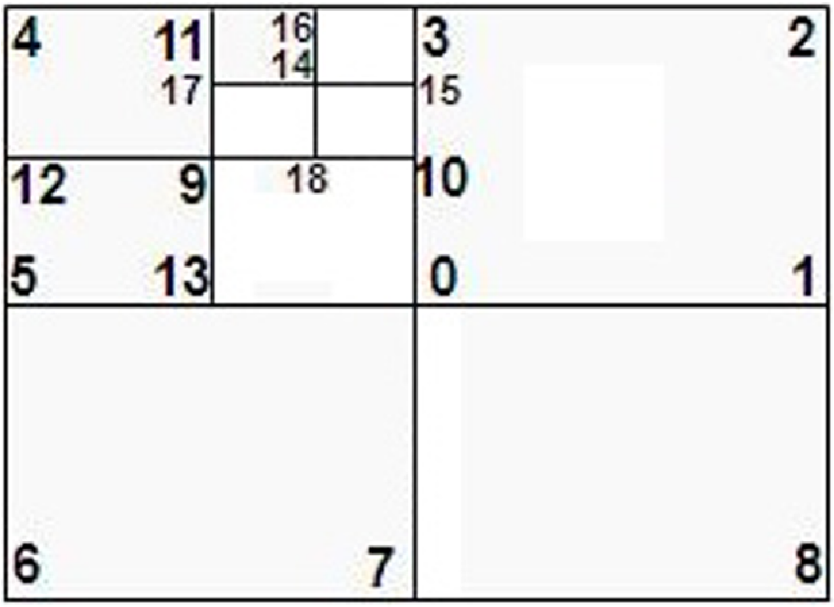


Fig. 3. Example of the vertice numbering by the adaptation of the MHC

node of the second linked list is allocated for this purpose whereas the pointers of the first linked list (which numbers the proper finite elements) are included in the proper graph nodes. In other words, there is a pointer in each graph node and these pointers implement the (first one) linked list to number the finite elements.

3.1 Assembling the Resulting Stiffness Matrix

The goal of those structures is to assemble the resulting stiffness matrix with the adequate connectivity information among neighbor elements. Furthermore, each line of the stiffness matrix is associated with a mesh vertice and is represented by a different simple linked list. In addition, since this scheme produces a sparse stiffness matrix, only non-zero values as well as their position are saved into the linked lists that represent the stiffness-matrix lines.

Notice that there are a linked list to number the finite elements, a linked list to number the vertices and each line of the resulting stiffness matrix is also a linked list.

Those linked lists that represent the stiffness-matrix lines are updated by a computational procedure that scans the vertex-linked list. It applies a linear interpolation of the associated corner-element values in order to solve the hanging nodes.

Numerical results show that this numbering scheme produces a sparse symmetric positive-definite matrix [2]. Moreover, the Conjugate Gradient Method was used in order to solve the resulting stiffness matrix.

4 Numerical Results

Firstly, results of two patch tests of the Laplace Problem is presented. Besides, the rule 2:1 was used in order to control the number of refinements. Secondly, a Poisson equation problem that involves steep gradients is solved. A 4×4 Gauss-Legendre Quadrature on four-node bilinear quadrilateral elements was applied. In the numerical tests, the maximum permissible error in the energy norm was set to five percent ($\eta_{max} = 0.05$).

4.1 AMR Strategy

Consider the elliptic boundary-value problem

$$-\nabla^2 u(x) = f(x) \text{ in } \Omega, \tag{1}$$

subject to the boundary conditions

$$u = 0 \text{ on } \partial\Omega . \tag{2}$$

The variational form of these equations can be written in the bilinear form

$$B(u,v) = \int \nabla u \nabla v d\Omega, \tag{3}$$

where $v \in H_0^1(\Omega)$.

Observe that $H_0^1(\Omega)$ is the Sobolev Space of functions with square-integrable derivatives and the values on $\partial\Omega$ vanish.

One can define the error as

$$error = u - u_h, \tag{4}$$

where u is the real solution and u_h is the numerical approximation. However, since this local measure is not computationally convenient, mathematical norms are introduced in order to measure the error. The energy error norm is related to the weak form in Eq. (3), thus,

$$||e||^2 = B(e,e) = \int \nabla e \nabla e d\Omega \ . \tag{5}$$

Since the exact solution field is generally unknown, the approximate solution is post-processed in order to obtain a more accurate measure for the gradient of u_h [3]. The relative percentage error in the energy norm is

$$\eta = \frac{||e||}{||u||}, \tag{6}$$

where $||u||$ is the exact energy norm.

A simple criterion in order to achieve a solution with an acceptable error is

$$\eta_{max} \leq \eta, \tag{7}$$

where η_{max} is the maximum permissible error percentage in the whole domain and

$$\eta = \frac{||e||}{(||u^h||^2 + ||e||^2)^{\frac{1}{2}}} \ . \tag{8}$$

The error should be equally distributed among the elements in order to obtain an optimal mesh

$$||e|| = \sqrt{m}||e||_i, \tag{9}$$

where m is the number of elements.

The error can be expressed for the entire domain using the admissible elementar error [4]

$$||e||_i \leq \eta_{max} \left(\frac{(||u^h||^2 + ||e||^2)}{m} \right)^{\frac{1}{2}} \equiv e_m^- \, . \tag{10}$$

Furthermore, if the error in an element is

$$\xi_i = \frac{||e||_i}{e_m^-} > 1, \tag{11}$$

the error in the element i is larger then the criterion and, therefore, it needs to be refined.

4.2 Validation

The Laplace problem

$$\nabla^2 u = 0, \Omega = [0,1]^2, \tag{12}$$

with $u = g(x) = x_1 + x_2$ over $\partial\Omega$ is solved.

The analyical solution is $u(x) = x_1 + x_2$. In this test, $\eta_{max} = 0.0$ was adopted in order to refine the elements. Figure 4 depicts an example of the accuracy of this test.

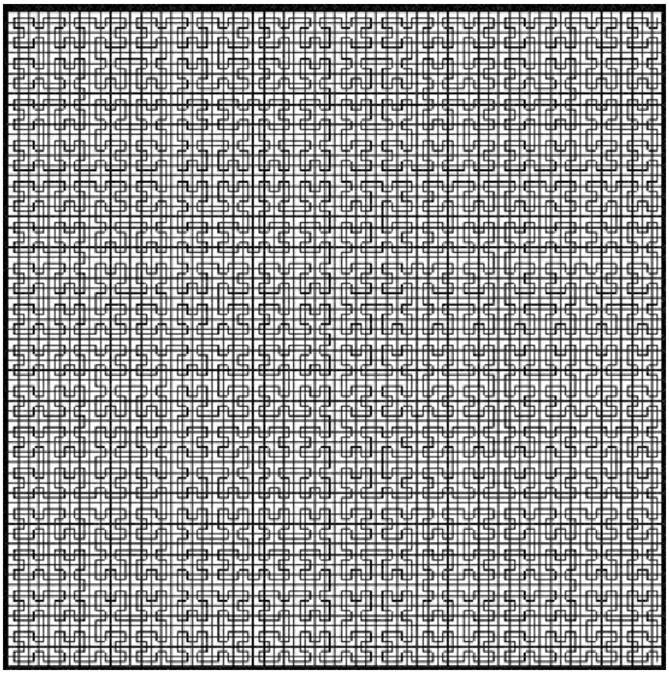


Fig. 4. Uniform refinement of the Laplace Problem comprising 1089 vertices

Table 1. Table 1: Laplace Problem results

Mesh	Number of elements	Number of vertices	Energy norm error
a	16	25	$1.7X10^{-17}$
b	256	289	$4.4X10^{-18}$
c	1024	1089	$2.2X10^{-18}$

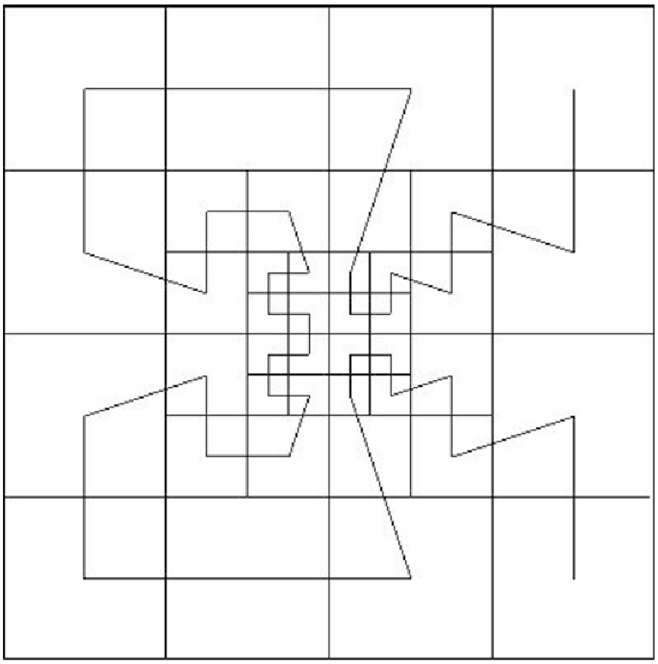


Fig. 5. Mesh of the Laplace Problem with 41 vertices and the associated MHC

Table 1 shows a comparison of the energy norm error among different meshes. A better numerical approximation when more elements are in the mesh is obtained.

Another test admitted 0.05 as the maximum error (η_{max}). Figure 5 shows an example mesh for this test. Fig. 6 presents a graphic of the vertice quantity by the error of the energy semi-norm H_1, showing the convergence of the solution.

4.3 Test in the Poisson Equation Problem

Consider the Poisson equation problem in $\Omega = [0,1]^2$ with Dirichlet boundary conditions

$$-\nabla^2 u = f(x,y) \ . \tag{13}$$

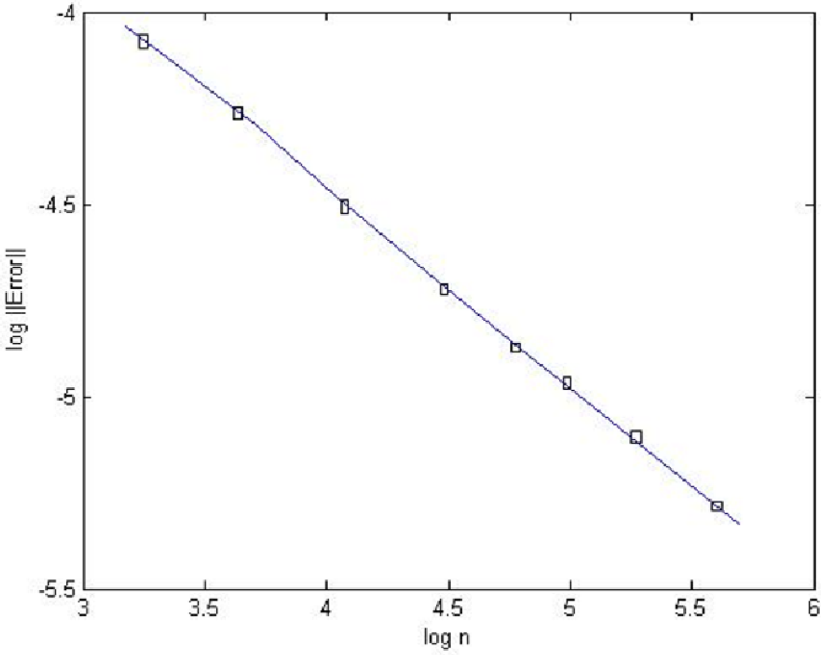


Fig. 6. Refinament convergence rate where n is the number of equations

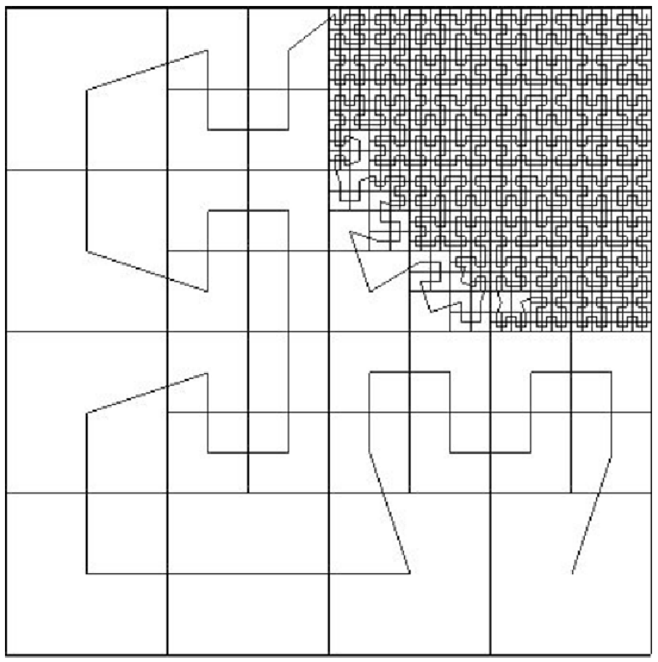


Fig. 7. Mesh of the Poisson equation problem with 248 elements

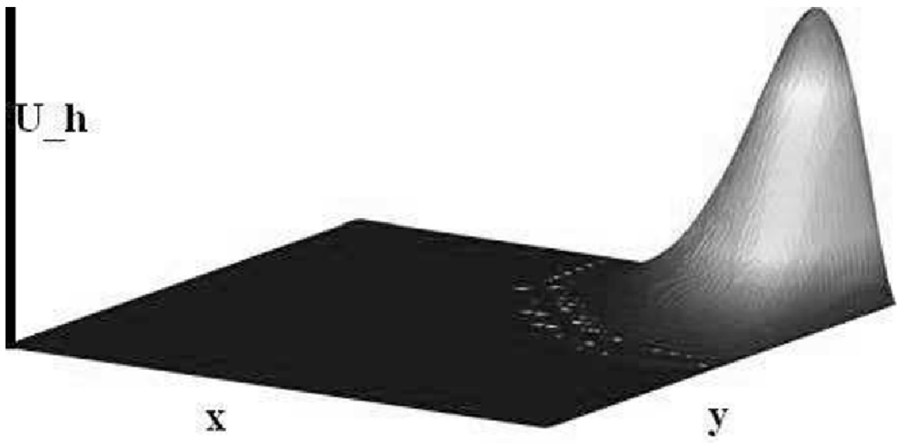


Fig. 8. Behavior of the solution u_h for the Poisson equation problem

with condition $u = 0$ over $\partial\Omega$.

The source term is given by

$$f(x, y) = -90x^8y^{10}(1-x)(1-y) + 20x^9y^{10}(1-y) - 90x^{10}y^8(1-x)(1-y) + 20y^9x^{10}(1-x)\ . \tag{14}$$

The analitical solution is $u(x) = x^{10}y^{10}(1-x)(1-y)$. Figure 7 shows a mesh for this test. Figure 8 shows that FEM-ALG represents the gradient close to the point $(1, 1)$.

5 Concluding Remarks

FEM-ALG is a novel technique for the adaptive mesh refinement that integrates the FEM and the ALG. This scheme intends to be more computationally efficient than other AMR schemes when searching for connectivity information among elements with different levels of refinement since the refinement process is merely local when updating all the involved structures.

This work shows experimental results of this scheme in the Poisson equation problem. More experiments shall be performed in the heating conduction equation and in the wave equation in future works. Moreover, future works will describe the unrefinement process. Besides, future works will describe the integration of new techniques in order to treat hanging nodes as well as the integration of mesh-free methods in order to apply the shape functions on nonconforming meshes.

Acknowledgements

The authors gratefully acknowledge CNPq, CAPES and FAPERJ for the financial support.

References

1. Burgarelli, D., Kischinhevsky, M., Biezuner, R.: A new adaptive mesh refinement strategy for numerically solving evolutionary PDE's. J. of Computational and Applied Mathematics. 196, 115–131 (2006)
2. Brandao, D.: A h-adaptative mesh refinement for the Finite Element Method using a graph structure (in Portuguese). Instituto de Computação da Universidade Federal Fluminense (2008)
3. Tabarraei, A., Sukumar, N.: Adaptive computations on conforming quadtree meshes. Finite Elements in Analysis and Design 41, 686–702 (2005)
4. Hughes, T.: The Finite Element Method. Englewood Cliffs (1987)

Integers Powers of Certain Asymmetric Matrices

Roman Wituła and Damian Słota

Institute of Mathematics
Silesian University of Technology
Kaszubska 23, 44-100 Gliwice, Poland
{roman.witula,damian.slota}@polsl.pl

Abstract. In this paper we derive the recurrent formulae for any integers powers of certain asymmetric matrices five order which covers also constans tridiagonal matrices and some asymmetric (and symmetric) pentadiagonal matrices. Since we find also the Binet's formulae for the elements of the powers of these matrices, it is possible to operate with powers series of these matrices. We note that this method is alternative to the Jordan method decomposition.

Keywords: tridiagonal matrices, asymmetric matrices.

1 Introduction

The scope of the paper is to determine the recurrent formulae for the powers of certain asymmetric matrices belonging to the class $M_{5\times 5}(\mathbb{C})$, involving constans tridiagonal matrices that have a very wide application. The recurrence method, which is an alternative to the Jordan's decomposition method, Leonard's algorithm, Putzer's algorithm and the method of characteristic polynomial [1,2,3,4,5,6,7], seems fast and facilitating the definition of integers powers of any matrices. The paper is a follow up on [8] (which is inspired by [9,10], see also [11]), where the cases of the third and fourth order asymmetric matrices are discussed and where also the complex powers of these matrices are defined. Apart from the proposed formulae, the authors also present a computational example of the application of this procedure.

2 Basic Recurrence Formulae

Let us set now:

$$\mathbf{G}_5(a,b,c,d,e,f,g,A,B,y) := \begin{bmatrix} a & y\,b & y^2\,c & y^3\,e & y^4\,g \\ b & A & y\,d & y^2\,f & y^3\,e \\ c & d & B & y\,d & y^2\,c \\ e & f & d & A & y\,b \\ g & e & c & b & a \end{bmatrix} \tag{1}$$

for any $a,b,c,d,e,f,g,A,B,y, \in \mathbb{C}$.

Lemma 1. *Let $a_1, b_1, c_1, d_1, e_1, f_1, g_1, A_1, B_1, y, \in \mathbb{C}$. If*

$$\begin{aligned} d_1 &= b_1 + y\, e_1, \\ f_1 &= c_1 + y\, g_1, \\ A_1 &= a_1 + y\, c_1, \\ B_1 &= a_1 + y\, c_1 + y^2\, g_1, \end{aligned}$$

then we get

$$\Big(\mathbf{G}_5(a_1, b_1, c_1, d_1, e_1, f_1, g_1, A_1, B_1, y)\Big)^k = \\ = \mathbf{G}_5(a_k, b_k, c_k, d_k, e_k, f_k, g_k, A_k, B_k, y) \quad (2)$$

for every $k \in \mathbb{N}$, where the following recurrence formulae hold:

$$\begin{aligned} a_{k+1} &= a_1\, a_k + y\, b_1\, b_k + y^2\, c_1\, c_k + y^3\, e_1\, e_k + y^4\, g_1\, g_k, && (3) \\ b_{k+1} &= b_1\, a_k + A_1\, b_k + y\, d_1\, c_k + y^2\, f_1\, e_k + y^3\, e_1\, g_k, && (4) \\ c_{k+1} &= c_1\, a_k + B_1\, c_k + y^2\, c_1\, g_k + d_1\, d_k \\ &= B_1\, c_k + B_k\, c_1 - y\, c_1\, c_k + d_1\, d_k, && (5) \\ e_{k+1} &= e_1\, a_k + f_1\, b_k + d_1\, c_k + A_1\, e_k + y\, b_1\, g_k, && (6) \\ g_{k+1} &= g_1\, a_k + e_1\, b_k + c_1\, c_k + b_1\, e_k + a_1\, g_k, && (7) \\ d_k &= b_k + y\, e_k, && (8) \\ f_k &= c_k + y\, g_k, && (9) \\ A_k &= a_k + y\, c_k, && (10) \\ B_k &= a_k + y\, c_k + y^2\, g_k. && (11) \end{aligned}$$

We note that if additionally we have $c_1 = g_1 = e_1 = 0$ then $G_5(a_1, \ldots, y)$ is a tridiagonal matrix.

The proof of the lemma by an easy induction arguments follows.

Now we want to find the Binet's formulae for the elements a_k, b_k, c_k, e_k and g_k. To this aim it is sufficient to designate twenty five coefficients ξ_i for $\xi \in \{\alpha, \beta, \gamma, \delta, \varepsilon\}$ and $i = 1, 2, 3, 4, 5$, for which the following system of equations hold:

$$\begin{aligned} a_k &= \alpha_1\, t^k + \beta_1\, u^k + \gamma_1\, v^k + \delta_1\, w^k + \varepsilon_1\, z^k, && (12) \\ y^{1/2}\, b_k &= \alpha_2\, t^k + \beta_2\, u^k + \gamma_2\, v^k + \delta_2\, w^k + \varepsilon_2\, z^k, && (13) \\ y\, c_k &= \alpha_3\, t^k + \beta_3\, u^k + \gamma_3\, v^k + \delta_3\, w^k + \varepsilon_3\, z^k, && (14) \\ y^{3/2}\, e_k &= \alpha_4\, t^k + \beta_4\, u^k + \gamma_4\, v^k + \delta_4\, w^k + \varepsilon_4\, z^k && (15) \\ y^2\, g_k &= \alpha_5\, t^k + \beta_5\, u^k + \gamma_5\, v^k + \delta_5\, w^k + \varepsilon_5\, z^k, && (16) \end{aligned}$$

for every $k \in \mathbb{N}$ and for some $t, u, v, w, z \in \mathbb{C}$ independent on k.

From identities (3)–(7) and (12)–(16) if we compare the coefficients of powers u^{k+1}, v^{k+1}, w^{k+1} and z^{k+1} it is easily established the following system of equations

$$\xi_1 = \xi_1^2 + \xi_2^2 + \xi_3^2 + \xi_4^2 + \xi_5^2, \tag{17}$$

$$\xi_2 = 2\,\xi_1\,\xi_2 + 2\,\xi_2\,\xi_3 + 2\,\xi_3\,\xi_4 + 2\,\xi_4\,\xi_5, \tag{18}$$

$$\xi_3 = 2\,\xi_1\,\xi_3 + 2\,\xi_2\,\xi_4 + 2\,\xi_3\,\xi_5 + \xi_2^2 + \xi_3^2 + \xi_4^2, \tag{19}$$

$$\xi_4 = 2\,\xi_1\,\xi_4 + 2\,\xi_2\,\xi_3 + 2\,\xi_3\,\xi_4 + 2\,\xi_2\,\xi_5, \tag{20}$$

$$\xi_5 = 2\,\xi_1\,\xi_5 + 2\,\xi_2\,\xi_4 + \xi_3^2, \tag{21}$$

for every $\xi \in \{\alpha, \beta, \gamma, \delta, \varepsilon\}$.
Sketch of solution of the system (17)–(21):

Substracting equations: (21) from (17) and next (18) from (20) we obtain respectively:

$$\xi_1 - \xi_5 = \big(\xi_1 - \xi_5\big)^2 + \big(\xi_2 - \xi_4\big)^2, \tag{22}$$

$$\xi_2 - \xi_4 = 2\,\xi_1\,\big(\xi_2 - \xi_4\big) + 2\,\xi_5\,\big(\xi_4 - \xi_2\big) = 2\,\big(\xi_2 - \xi_4\big)\,\big(\xi_1 - \xi_5\big),$$

i.e.,

$$\big(\xi_2 - \xi_4\big)\,\big(1 - 2\,\big(\xi_1 - \xi_5\big)\big) = 0. \tag{23}$$

From (23) we have, that either $\xi_2 - \xi_4 = 0$ or $\xi_1 - \xi_5 = \frac{1}{2}$. If $\xi_2 = \xi_4$ then by (22) either $\xi_1 = \xi_5$ or $\xi_1 - \xi_5 = 1$. On the other hand, if $\xi_1 - \xi_5 = \frac{1}{2}$, then by (22) $\xi_2 - \xi_4 = \pm\frac{1}{2}$. Finally, the equations system (17)–(21) is extended to include one of the following four equations:

1. $\xi_2 = \xi_4$ and $\xi_1 = \xi_5$,
2. $\xi_2 = \xi_4$ and $\xi_1 - \xi_5 = 1$,
3. $\xi_1 - \xi_5 = \frac{1}{2}$ and $\xi_2 - \xi_4 = \frac{1}{2}$,
4. $\xi_1 - \xi_5 = \frac{1}{2}$ and $\xi_2 - \xi_4 = -\frac{1}{2}$,

reducing the number of solutions to 28 vectors from $\mathbb{R}^5$ (all calculations were performed in *Mathematica*). Surely, out of the 28 vectors, we are interested in only five different ones. It should be indicated that in view of equations (3)–(7), the required vectors must comply with the following orthogonal conditions:

$$\sum_{i=1}^{5} \zeta_i\,\eta_i = \sum_{i=1}^{5} \zeta_i\,\eta_{5-i} = 0$$

for every $\zeta, \eta \in \{\alpha, \beta, \gamma, \delta, \varepsilon\}$, $\zeta \neq \eta$. From this point, it is easy to derive the following solution:

$$\begin{bmatrix} \alpha_1 & \beta_1 & \gamma_1 & \delta_1 & \varepsilon_1 \\ \alpha_2 & \beta_2 & \gamma_2 & \delta_2 & \varepsilon_2 \\ \alpha_3 & \beta_3 & \gamma_3 & \delta_3 & \varepsilon_3 \\ \alpha_4 & \beta_4 & \gamma_4 & \delta_4 & \varepsilon_4 \\ \alpha_5 & \beta_5 & \gamma_5 & \delta_5 & \varepsilon_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{12} & \frac{1}{4} & \frac{1}{3} & \frac{1}{4} & \frac{1}{12} \\ \frac{-1}{4\sqrt{3}} & \frac{1}{4} & 0 & -\frac{1}{4} & \frac{1}{4\sqrt{3}} \\ \frac{1}{6} & 0 & -\frac{1}{3} & 0 & \frac{1}{6} \\ \frac{-1}{4\sqrt{3}} & -\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4\sqrt{3}} \\ \frac{1}{12} & -\frac{1}{4} & \frac{1}{3} & -\frac{1}{4} & \frac{1}{12} \end{bmatrix}. \tag{24}$$

Moreover, we can generate the following formulae

$$t = a_1 - b_1\sqrt{3\,y} + 2\,c_1\,y + g_1\,y^2 - e_1\sqrt{3\,y^3}, \tag{25}$$

$$u = a_1 + b_1\sqrt{y} - g_1\,y^2 - e_1\sqrt{y^3}, \tag{26}$$

$$v = a_1 - c_1\,y + g_1\,y^2, \tag{27}$$

$$w = a_1 - b_1\sqrt{y} - g_1\,y^2 + e_1\sqrt{y^3}, \tag{28}$$

$$z = a_1 + b_1\sqrt{3\,y} + 2\,c_1\,y + g_1\,y^2 + e_1\sqrt{3\,y^3}. \tag{29}$$

Corollary 1. *We have*

$$\begin{bmatrix} a & c & 0 & 0 & 0 \\ b & a & c & 0 & 0 \\ 0 & b & a & c & 0 \\ 0 & 0 & b & a & c \\ 0 & 0 & 0 & b & a \end{bmatrix}^r = \mathbf{G}_5\big(\alpha_r, \beta_r, \gamma_r, \delta_r, \varepsilon_r, f_r, g_r, A_r, B_r, \frac{c}{b}\big), \tag{30}$$

where

$$\alpha_r = \frac{1}{12}\,t^r + \frac{1}{4}\,u^r + \frac{1}{3}\,v^r + \frac{1}{4}\,w^r + \frac{1}{12}\,z^r, \tag{31}$$

$$y^{1/2}\,\beta_r = \frac{-1}{4\sqrt{3}}\,t^r + \frac{1}{4}\,u^r - \frac{1}{4}\,w^r + \frac{1}{4\sqrt{3}}\,z^r, \tag{32}$$

$$y\,\gamma_r = \frac{1}{6}\,t^r - \frac{1}{3}\,v^r + \frac{1}{6}\,z^r, \tag{33}$$

$$y^{3/2}\,\varepsilon_r = \frac{-1}{4\sqrt{3}}\,t^r - \frac{1}{4}\,u^r + \frac{1}{4}\,w^r + \frac{1}{4\sqrt{3}}\,z^r, \tag{34}$$

$$y^2\,g_r = \frac{1}{12}\,t^r - \frac{1}{4}\,u^r + \frac{1}{3}\,v^r - \frac{1}{4}\,w^r + \frac{1}{12}\,z^r, \tag{35}$$

$$y^{1/2}\,\delta_r = y^{1/2}\,\beta_r + y^{3/2}\,\varepsilon_r = \frac{\sqrt{3}}{6}\,\big(z^r - t^r\big), \tag{36}$$

$$4\,y\,f_r = 4\,\big(y\,\gamma_r + y^2\,g_r\big) = t^r - u^r - w^r + z^r, \tag{37}$$

$$4\,A_r = 4\,\big(\alpha_r + y\,\gamma_r\big) = t^r + u^r + w^r + z^r, \tag{38}$$

$$3\,B_r = 3\,\big(\alpha_r + y\,\gamma_r + y^2\,g_r\big) = t^r + v^r + z^r, \tag{39}$$

and

$$\begin{aligned} t &= a - \sqrt{3\,b\,c}, \\ u &= a + \sqrt{b\,c}, \\ v &= a, \\ w &= a - \sqrt{b\,c}, \\ z &= a + \sqrt{3\,b\,c}. \end{aligned}$$

Remark 1. All formulas (2)–(16) and (30)–(39) hold true also for all $k, r \in \mathbb{Z}$, since $\mathbf{G}_5\big(a_1, b_1, \ldots, g_1, A_1, B_1, y\big)$ is the diagonal matrix (see also [8] for more comments). The powers in the respective formulas should be chosen in the following way:

$$x^r := \exp\bigl(r\,(\ln|x| + i\,\mathrm{Arg}\,x)\bigr).$$

3 Example

Now the numerical example of application of our formulae will be given. Let us set: $a_1 = 1$, $b_1 = 1$, $c_1 = -1$, $e_1 = -1$, $g_1 = 1$, $d_1 = 1 - i = \sqrt{2}\,e^{-i\,\pi/4}$, $f_1 = -1 + i = \sqrt{2}\,e^{i\,3\,\pi/4}$, $A_1 = 1 - i = \sqrt{2}\,e^{-i\,\pi/4}$, $B_1 = -i$, $y = i$. For these values, we have

$$\mathbf{G}_5(a_1, b_1, \ldots, B_1, y) = \begin{bmatrix} 1 & i & 1 & i & 1 \\ 1 & 1-i & 1+i & 1-i & i \\ -1 & 1-i & -i & 1+i & 1 \\ -1 & -1+i & 1-i & 1-i & i \\ 1 & -1 & -1 & 1 & 1 \end{bmatrix}.$$

By (25)–(29) we get

$$\begin{aligned} t &= 1 - \sqrt{3\,i} - 2\,i - 1 + \sqrt{-3\,i} \\ &= -\exp\bigl(\tfrac{1}{2}(\ln 3 + i\,\tfrac{\pi}{2})\bigr) - 2\,i + \exp\bigl(\tfrac{1}{2}(\ln 3 + i\,\tfrac{3\pi}{2})\bigr) \\ &= -\sqrt{3}\,\exp\bigl(i\,\tfrac{\pi}{4}\bigr) - 2\,i + \sqrt{3}\,\exp\bigl(i\,\tfrac{3\pi}{4}\bigr) \\ &= -\sqrt{6} - 2\,i, \\ u &= 2 + i\,\sqrt{2}, \\ v &= i, \\ w &= 2 - i\,\sqrt{2}, \\ z &= \sqrt{6} - 2\,i. \end{aligned}$$

Then by (2) we have

$$\mathbf{G}_5^k(a_1, b_1, \ldots, B_1, y) = \mathbf{G}_5(a_k, b_k, \ldots, B_k, y), \qquad k \in \mathbb{Z},$$

where

$$\begin{aligned} a_k &= \frac{1}{12}\bigl(-(\sqrt{6} + 2\,i)\bigr)^k + \frac{1}{4}\bigl(2 + i\,\sqrt{2}\bigr)^k + \frac{1}{3}\,i^k + \frac{1}{4}\bigl(2 - i\,\sqrt{2}\bigr)^k \\ &\quad + \frac{1}{12}\bigl(\sqrt{6} - 2\,i\bigr)^k, \\ e^{i\,\pi/4}\,b_k &= -\frac{\sqrt{3}}{12}\bigl(-(\sqrt{6} + 2\,i)\bigr)^k + \frac{1}{4}\bigl(2 + i\,\sqrt{2}\bigr)^k - \frac{1}{4}\bigl(2 - i\,\sqrt{2}\bigr)^k \\ &\quad + \frac{\sqrt{3}}{12}\bigl(\sqrt{6} - 2\,i\bigr)^k, \\ i\,c_k &= \frac{1}{6}\bigl(-(\sqrt{6} + 2\,i)\bigr)^k - \frac{1}{3}\,i^k + \frac{1}{6}\bigl(\sqrt{6} - 2\,i\bigr)^k, \end{aligned}$$

$$\begin{aligned}
e^{-i\,\pi/4}\,e_k &= -\frac{\sqrt{3}}{12}\left(-(\sqrt{6}+2\,i)\right)^k - \frac{1}{4}\left(2+i\,\sqrt{2}\right)^k + \frac{1}{4}\left(2-i\,\sqrt{2}\right)^k \\
&\quad + \frac{\sqrt{3}}{12}\left(\sqrt{6}-2\,i\right)^k, \\
g_k &= -\frac{1}{12}\left(-(\sqrt{6}+2\,i)\right)^k + \frac{1}{4}\left(2+i\,\sqrt{2}\right)^k - \frac{1}{3}\,i^k + \frac{1}{4}\left(2-i\,\sqrt{2}\right)^k \\
&\quad - \frac{1}{12}\left(\sqrt{6}-2\,i\right)^k, \\
2\,e^{i\,\pi/4}\,d_k &= \left(2+i\,\sqrt{2}\right)^k - \left(2-i\,\sqrt{2}\right)^k, \\
4\,i\,f_k &= \left(-(\sqrt{6}+2\,i)\right)^k - \left(2+i\,\sqrt{2}\right)^k - \left(2-i\,\sqrt{2}\right)^k + \left(\sqrt{6}-2\,i\right)^k, \\
4\,A_k &= \left(-(\sqrt{6}+2\,i)\right)^k + \left(2+i\,\sqrt{2}\right)^k + \left(2-i\,\sqrt{2}\right)^k + \left(\sqrt{6}-2\,i\right)^k, \\
3\,B_k &= \left(-(\sqrt{6}+2\,i)\right)^k + i^k + \left(\sqrt{6}-2\,i\right)^k.
\end{aligned}$$

For example, for $k=-1$ we get

$$\mathbf{G}_5^{-1}(a_1,\ldots,B_1,y) = \begin{bmatrix}
\frac{1}{6}-\frac{3i}{10} & \frac{2}{15}-\frac{i}{30} & -\frac{2}{5} & -\frac{2}{15}-\frac{i}{30} & \frac{1}{6}+\frac{3i}{10} \\
-\frac{1}{30}-\frac{2i}{15} & \frac{1}{6}+\frac{i}{10} & \frac{1}{10}+\frac{i}{10} & -\frac{1}{10}-\frac{i}{6} & -\frac{2}{15}-\frac{i}{30} \\
\frac{2}{5} & \frac{1}{10}-\frac{i}{10} & -\frac{i}{5} & \frac{1}{10}+\frac{i}{10} & -\frac{2}{5} \\
\frac{1}{30}-\frac{2i}{15} & \frac{1}{10}+\frac{i}{6} & \frac{1}{10}-\frac{i}{10} & \frac{1}{6}+\frac{i}{10} & \frac{2}{15}-\frac{i}{30} \\
\frac{1}{6}+\frac{3i}{10} & \frac{1}{30}-\frac{2i}{15} & \frac{2}{5} & -\frac{1}{30}-\frac{2i}{15} & \frac{1}{6}-\frac{3i}{10}
\end{bmatrix}.$$

4 Final Remarks

1° It's some natural problem how to defined some power series of $\mathbf{G}_5$. For example, how $e^{\mathbf{G}_5}$ should be defined? By Lemma 1 it can be defined in the following way:

$$\begin{aligned}
&\exp\left(\mathbf{G}_5(a,b,c,d,e,f,g,A,B,y)\right) \approx \\
&\qquad \approx \sum_{k=0}^{N} \frac{1}{k!}\,\mathbf{G}_5(a_k,b_k,c_k,d_k,e_k,f_k,g_k,A_k,B_k,y) = \\
&\qquad\qquad = \mathbf{G}_5\Big(\sum_{k=0}^{N}\frac{1}{k!}\,a_k, \sum_{k=0}^{N}\frac{1}{k!}\,b_k,\ldots,\sum_{k=0}^{N}\frac{1}{k!}\,B_k, y\Big),
\end{aligned}$$

where N can be chosen arbitrary with a given accuracy, because of the Binet's formulae (12)–(16). On the other hand, we have the following exact formula:

$$\exp\Big(\mathbf{G}_5(a,b,c,d,e,f,g,A,B,y)\Big) = \mathbf{G}_5(\widehat{a},\widehat{b},\widehat{c},\widehat{d},\widehat{e},\widehat{f},\widehat{g},\widehat{A},\widehat{B},y),$$

where

$$\begin{bmatrix} \widehat{a} \\ y^{1/2}\,\widehat{b} \\ y\,\widehat{c} \\ y^{3/2}\,\widehat{e} \\ y^2\,\widehat{g} \end{bmatrix} = \begin{bmatrix} \frac{1}{12} & \frac{1}{4} & \frac{1}{3} & \frac{1}{4} & \frac{1}{12} \\ \frac{-1}{4\sqrt{3}} & \frac{1}{4} & 0 & -\frac{1}{4} & \frac{1}{4\sqrt{3}} \\ \frac{1}{6} & 0 & -\frac{1}{3} & 0 & \frac{1}{6} \\ \frac{-1}{4\sqrt{3}} & -\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4\sqrt{3}} \\ \frac{1}{12} & -\frac{1}{4} & \frac{1}{3} & -\frac{1}{4} & \frac{1}{12} \end{bmatrix} \begin{bmatrix} e^t \\ e^u \\ e^v \\ e^w \\ e^z \end{bmatrix},$$

for the definition of the parameters t, u, v, w, z see formulae (25)–(29), and

$$\widehat{d} = \widehat{b} + y\,\widehat{e},$$
$$\widehat{f} = \widehat{c} + y\,\widehat{g},$$
$$\widehat{A} = \widehat{a} + y\,\widehat{c},$$
$$\widehat{B} = \widehat{a} + y\,\widehat{c} + y^2\,\widehat{g}.$$

In the connection with these deliberations it is intriguing in some sense that *Mathematica* have some problem with symbolic Jordan decomposition of the matrix $\mathbf{G}_5(a_1, b_1, \ldots, B_1, y)$ from our example.

2° Why are only the matrices 5×5 discussed in the paper? It turned out that generalizations of the case 3×3 (from [8]) onto 4×4 and 5×5 matrices are unique in a certain sense. The case of the 6×6 matrix:

$$\mathbf{G}_6 := \begin{bmatrix} a & y\,b & y^2\,c & y^3\,e & y^4\,g & y^5\,k \\ b & A & y\,d & y^2\,f & y^3\,h & y^4\,g \\ c & d & B & y\,d & y^2\,f & y^3\,e \\ e & f & d & B & y\,d & y^2\,c \\ g & h & f & d & A & y\,b \\ k & g & e & c & b & a \end{bmatrix}$$

with respective recurrence relations for its elements don't work. In consequence the generalization $n \times n$, $n \geq 6$, are not so strong (it is necessary to use less number of parameters or others recurrence relations).

References

1. Abu-Saris, R., Ahmad, W.: Avoiding eigenvalues in computing matrix power. Amer. Math. Monthly 112, 450–454 (2005)
2. Barakat, R., Baumann, E.: Mth power on $N \times N$ matrix and its connection with the generalized Lucas polynomial. J. Math. Phys. 10, 1474–1476 (1969)
3. Beckenbach, E.F.: Modern Mathematics for the Engineer. McGraw-Hill, New York (1961)
4. Elaydi, S.N., Harris Jr., W.A.: On the computation of A^N. SIAM Rev. 40, 965–971 (1998)
5. Fulmer, E.P.: Computation of the matrix exponential. Amer. Math. Monthly 82, 156–159 (1975)
6. Leonard, I.E.: The matrix exponential. SIAM Rev. 38, 507–512 (1996)

7. Putzer, E.J.: Avoiding the Jordan canonical form in the discussion of linear systems with constant coefficients. Amer. Math. Monthly 73, 2–7 (1966)
8. Wituła, R., Słota, D.: Some phenomenon of the powers of certain tridiagonal and asymmetric matrices. Appl. Math. Comput. 202, 348–359 (2008)
9. Samarskii, A.A., Gulin, A.V.: The Stability of the Difference Schemes. Nauka, Moscow (in Russian) (1973)
10. Wituła, R., Słota, D.: On computing the determinants and inverses of some special type of tridiagonal and constant-diagonals matrices. Appl. Math. Comput. 189, 514–527 (2007)
11. Kilic, E.: On a constant-diagonals matrix. Appl. Math. Comput. 204, 184–190 (2008)

Short Papers

Numerical Simulation of the Dynamics of a Periodically Forced Spherical Particle in a Quiescent Newtonian Fluid at Low Reynolds Numbers

Tumkur Ramaswamy Ramamohan[1], Inapura Siddagangaiah Shivakumara[2], and Krishnamurthy Madhukar[1,2]

[1] Centre for Mathematical Modelling and Computer Simulation (C-MMACS), Council of Scientific and Industrial Research, Wind Tunnel Road, Bangalore – 560 037, India
[2] UGC-Centre for Advanced Studies in Fluid Mechanics, Department of Mathematics, Bangalore University, Bangalore – 560 001, India
trr@cmmacs.ernet.in, isshivakumara@hotmail.com, madhukar@cmmacs.ernet.in

Abstract. In this paper we present the results of a numerical simulation of the dynamics of a periodically forced spherical particle in a quiescent Newtonian fluid at low Reynolds number. We describe the simulation and tests performed to validate our simulation. We have obtained results which are physically reasonable and hence we have confidence in our results. We include the effects of both convective and unsteady inertia on the dynamics at low Reynolds numbers. The inclusion of inertia results in additional linear and nonlinear terms in the equations representing a fading memory of the entire history of the motion. The nonlinearity though small in the parametric regime of interest, gives rise to some interesting features in the solution of the problem.

Keywords: Low Reynolds numbers, quiescent fluid, spherical particle, periodic force.

1 Introduction

The motion of a spherical particle in a fluid at low Reynolds numbers has been of appreciable interest for more than a century, starting with Stokes [1], and his interest on the effects of fluid friction on the motion of pendulums and hence in accurate time keeping. His fundamental expression for the force acting on a spherical particle has motivated many researchers to obtain better approximations for the hydrodynamic force acting on spherical particles. Lovalenti and Brady [2] have summarized work prior to 1993 in their paper and have also derived an expression for the hydrodynamic force undergoing arbitrary time-dependent motion at small Reynolds numbers. In this paper we use the expression derived by Lovalenti and Brady [2] for the hydrodynamic force on a rigid spherical particle undergoing arbitrary time-dependent motion at low Reynolds numbers, to obtain expressions for the particle displacement and particle velocity of a periodically forced spherical particle in a quiescent Newtonian fluid at low Reynolds numbers.

Our work is motivated by the prior work of Ramamohan and coworkers on the dynamics of periodically forced particles at zero Reynolds numbers i.e., by complete neglect of both unsteady and convective inertia. Kumar and Ramamohan [3] have shown that rheological parameters of suspensions can be controlled using periodic perturbations and that small changes in controllable parameters lead to large changes in rheology. In the present work, our interest is in studying periodically forced suspensions at low Reynolds numbers. The present system is one of the simplest experimentally realizable fluid dynamical systems, at low Reynolds numbers, which can atleast in principle show nonlinear behavior. Further this is an ideal system to investigate fundamental questions about the average behavior of periodically forced systems of nonlinear oscillators and the relationship of the dynamics of a single oscillator to the dynamics of the average behavior of a large number of such oscillators. Here we note that the periodic forcing occurs both at the individual particle level as well as at the level of the averages [4]. This provides a nonlinear coupling between the microlevel of the individual particle and the macrolevel of its averages. In this work we take the first step in the direction of analyzing this problem by determining the effects of a periodic force on the dynamics of a neutrally buoyant spherical particle in a quiescent incompressible Newtonian fluid at low but nonzero Reynolds numbers. This is the simplest possible extension including inertia to the problem studied for over a decade by Ramamohan and coworkers and summarized in Asokan et al. [5]. The inclusion of inertia results in a delay between the variation of the external force and the variation in the response of the particle to the external force. The expression for the hydrodynamic force F^H on a spherical particle undergoing an arbitrary motion in an arbitrary time dependent uniform flow field, given by Lovalenti and Brady [2] is used to derive the expression for the particle velocity and displacement using Newton's law and the results are validated by performing several tests on the software. We have generated a number of phase plots (plots between particle displacement and particle velocity) and displacement time series and velocity time series from our simulation. We have analyzed these phase plots and time series using the TISEAN software package, Hegger et al, [6] for nonlinear behavior.

2 Formulation of the Problem

The Lovalenti and Brady [2] formalism for the hydrodynamic force on a rigid sphere undergoing arbitrary time – dependent motion in an arbitrary time dependent uniform flow field at small Reynolds numbers is given by the following expression:

$$
\begin{aligned}
F^H(t) = {} & \frac{4\pi}{3}\mathrm{Re}\; Sl\, \dot{U}^{\infty}(t) - 6\pi U_s(t) - \frac{2\pi}{3}\mathrm{Re}\; Sl\, \dot{U}_s(t) \\
& + \frac{3}{8}\left(\frac{\mathrm{Re}\; Sl}{\pi}\right)^{1/2}\left\{\int_{-\infty}^{t}\left[\frac{2}{3}F_s^{H_\parallel}(t) - \left\{\frac{1}{|A|^2}\left(\frac{\pi^{1/2}}{2|A|}erf\,(|A|) - \exp\left(-|A|^2\right)\right)\right\}F_s^{H_\parallel}(s)\right.\right. \\
& \left. + \frac{2}{3}F_s^{H_\perp}(t) - \left\{\exp\left(-|A|^2\right) - \frac{1}{2|A|^2}\left(\frac{\pi^{1/2}}{2|A|}erf\,(|A|) - \exp\left(-|A|^2\right)\right)\right\}F_s^{H_\perp}\right] \\
& \left. \times \frac{2\,ds}{(t-s)^{3/2}}\right\} + o(\mathrm{Re}).
\end{aligned}
\tag{1}
$$

This expression is obtained by using the reciprocal theorem and the details of the derivation can be found in Lovalenti and Brady [2]. Here, $U_s = U_p - U^\infty$ is the slip velocity of the fluid. U_p is the velocity of the particle. U^∞ is the velocity of the fluid as $|r| \to \infty$. Re is the Reynolds number, defined as $Re = aU_c/\nu$ based on a characteristic particle slip velocity, U_c, a denotes the characteristic particle dimension and ν is the kinematic viscosity of the fluid. $F_s^{H_\parallel} = -6\pi U_s . \underline{p}\,\underline{p}$ and $F_s^{H_\perp} = -6\pi U_s .(\delta - \underline{p}\,\underline{p})$, where δ is the idem tensor of order 2 and unit vector $\underline{p} = \dfrac{Y_s(t) - Y_s(s)}{|Y_s(t) - Y_s(s)|}$, here $Y_s(t) - Y_s(s)$ is the integrated displacement of the particle relative to the fluid from time s to the current time t. Sl is the Strouhal number and $A = \dfrac{\text{Re}}{2}\left(\dfrac{t-s}{\text{Re}\,Sl}\right)^{1/2}\left(\dfrac{Y_s(t)-Y_s(s)}{t-s}\right)$.

For our problem, we consider a neutrally buoyant sphere in an infinite body of a quiescent fluid and consider the effects of an external periodic force acting on the sphere along the x – axis. We use equation (1) to obtain the equation governing the unidirectional motion of a sphere in a quiescent fluid, starting with zero velocity at time t = 0, with $U_s = U_p$ where U_p is the velocity of the particle, scaled with respect to the size of the particle and the frequency of the external periodic force, ω and $U^\infty = 0$. Under these conditions, equation (1) reduces to

$$F^H(t) = -6\pi U_p(t) - \frac{2\pi}{3}\text{Re}\,Sl\dot{U}_p(t) + \frac{3}{8}\left(\frac{\text{Re}\,Sl}{\pi}\right)^{1/2}\int_0^t\left[\frac{-8\pi U_p(t)ds}{(t-s)^{3/2}} - \left\{\frac{1}{|A|^2}\left(\frac{\pi^{1/2}}{2|A|}erf(|A|) - \exp(-|A|^2)\right)\right\}\frac{-12\pi U_p(s)ds}{(t-s)^{3/2}}\right] \quad (2)$$

We note that the integral in equation (2) contains a singularity at $s = t$. In order to take account of this singularity, the integral was split into the intervals $[0, t-\varepsilon]$ and $[t-\varepsilon, t]$ for a small positive ε. We then transform the integral in the interval $[t-\varepsilon, t]$ with respect to A, where

$$A = \frac{\text{Re}}{2}\left(\frac{t-s}{\text{Re}\,Sl}\right)^{1/2}\left(\frac{Y_s(t)-Y_s(s)}{t-s}\right)$$

That is, we get,

$$F^H(t) = -6\pi U_p(t) - \frac{2\pi}{3}\text{Re}\,Sl\dot{U}_p(t) + \frac{3}{8}\left(\frac{\text{Re}\,Sl}{\pi}\right)^{1/2}(P+Q) \quad (3)$$

where

$$P=\int_0^{t-\varepsilon}\left[\frac{-8\pi U_p(t)ds}{(t-s)^{3/2}}-\left\{\frac{1}{|A|^2}\left(\frac{\pi^{1/2}}{2|A|}erf(|A|)-\exp\left(-|A|^2\right)\right)\right\}\frac{-12\pi U_p(s)ds}{(t-s)^{3/2}}\right] \tag{3a}$$

and

$$Q=\int_{t-\varepsilon}^{t}\left[\frac{-8\pi U_p(t)ds}{(t-s)^{3/2}}-\left\{\frac{1}{|A|^2}\left(\frac{\pi^{1/2}}{2|A|}erf(|A|)-\exp\left(-|A|^2\right)\right)\right\}\frac{-12\pi U_p(s)ds}{(t-s)^{3/2}}\right] \tag{3b}$$

Transforming the integral with respect to *A*, we get

$$Q=\int_0^{c\sqrt{\varepsilon}}\frac{8\pi U_p^2(t)\operatorname{Re}dA}{(\operatorname{Re}Sl)^{1/2}A^2}-\int_0^{c\sqrt{\varepsilon}}\frac{1}{|A^2|}\left(\frac{\sqrt{\pi}}{2|A|}erf(|A|)-\exp\left(-|A^2|\right)\right)\frac{12\pi U_p^2(s)\operatorname{Re}dA}{(\operatorname{Re}Sl)^{1/2}A^2} \tag{3c}$$

where $c=\frac{\operatorname{Re}U_p(t)}{2\sqrt{\operatorname{Re}Sl}}$. We note that, *Q* vanishes as ε tends to zero; i.e., as $s\to t$.

$$\frac{1}{|A^2|}\left(\frac{\sqrt{\pi}}{2|A|}erf(|A|)-\exp\left(-|A^2|\right)\right)\to\frac{2}{3}$$

Hence the two singular terms cancel each other as $s\to t$, and thus we obtain an expression for the hydrodynamic force on a sphere in a quiescent fluid as:

$$\begin{aligned}F^H(t)&=-6\pi U_p(t)-\frac{2\pi}{3}\operatorname{Re}Sl\dot{U}_p(t)\\&+\frac{3}{8}\left(\frac{\operatorname{Re}Sl}{\pi}\right)^{1/2}\left\{\int_0^{t-\varepsilon}\left[\left\{\frac{1}{|A|^2}\left(\frac{\pi^{1/2}}{2|A|}erf(|A|)-\exp\left(-|A|^2\right)\right)\right\}\frac{12\pi U_p(s)ds}{(t-s)^{3/2}}\right]\right.\\&\left.+16\pi U_p(t)\left[\frac{1}{\sqrt{t}}-\frac{1}{\sqrt{\varepsilon}}\right]\right\}\end{aligned} \tag{4}$$

The equation of motion for a neutrally buoyant particle immersed in a fluid is given by

$$\frac{m_p\dot{U}_p(t)}{\mu a^2\omega}=F^{ext}(t)+F^H(t), \tag{5}$$

Using equation (4) with the external periodic force $F^{ext}=F_0\ sin(t)$, where time has been scaled with respect to the frequency of the external periodic force field, along the x direction and using Newton's law, we obtain equations for the particle velocity U_p and position Y_p with velocity and position equal to zero at time equal to zero in the form

$$\frac{dY_p}{dt} = U_p \tag{5a}$$

$$\frac{dU_p}{dt} = \frac{1}{\mathrm{Re}^*}\left[\mathrm{Re}_F \sin(t) - 6\pi U_p + \frac{3}{8}\left(\frac{\mathrm{Re}\,Sl}{\pi}\right)^{1/2}(J_1 + I_1)\right] \tag{5b}$$

Here, $\mathrm{Re}^* = \frac{4\pi}{3}\mathrm{Re} + \frac{2\pi}{3}\mathrm{Re}\ Sl$, $\mathrm{Re}_F = \frac{F_0}{\mu a^2 \omega}$, $\mathrm{Re} = \frac{\rho a^2 \omega}{\mu}$, where, a is the particle size, ω is the frequency of the applied external periodic force, μ is the viscosity of the fluid and ρ is the density of the particle and the fluid, since the particle is assumed to be neutrally buoyant, we have

$$J_1 = 16\pi U_p(t)\left[\frac{1}{\sqrt{t}} - \frac{1}{\sqrt{\varepsilon}}\right]$$

and

$$I_1 = \int_0^{t-\varepsilon}\left\{\frac{1}{A^2}\left(\frac{\sqrt{\pi}}{2|A|}erf(A) - \exp(-A^2)\right)\right\}\frac{12\pi U_p(s)}{(t-s)^{3/2}}ds\,.$$

3 Methodology

We developed software using Numerical Recipes in FORTRAN [7] to solve equations (5a) and (5b) using an embedded Runge-Kutta method with adaptive step size. The integral in the equation (5b) was evaluated at each time step by Romberg extrapolation. The function with respect to ‘A’ was defined by a user supplied function subprogram. We used the ODEINT, RKQS, RKCK subroutines from Numerical Recipes [7] to implement the Runge – Kutta method. The Romberg extrapolation was performed using the QROMB subroutine. The integral was evaluated using TRAPZD and the interpolation during the numerical quadrature was performed by POLINT. The tolerance for both the Romberg extrapolation and the Runge - Kutta solver was taken as 10^{-5}. Further reduction of the tolerance did not result in any significant change in our results. The entire program was written in double precision. The initial conditions for both the velocity and the position of the particle were taken as zero. ε was taken as 0.04; smaller values of ε did not significantly change the results. The software was tested for consistency by compiling the program with two compilers namely, Intel Fortran and F90. We generated 5,000 data points taken at an interval of $\pi/200$ in both the dimensionless velocity and dimensionless position. We note that there are certain novel features of these equations, namely a nonlinear history term that results in certain special features in the solutions.

The results of both the programs matched except for minor differences of the order of 10^{-5}. In addition to this, we performed the following tests to check the validity of our results.

TEST1: We performed a perturbation analysis of the problem, using $\mathrm{Re}^{1/2}$ as the perturbation parameter. The particle displacement and particle velocity matched with our numerical solutions upto $\mathrm{Re} \approx 0.05$. The deviation from the perturbation solution may be due to the fact that even though Re is small, 'A' in equation (4), need not be small for all 't' and 's'. This test is an important test on our software as it showed that the numerical solutions obtained by our software are correct.

TEST2: The solution of the problem of the motion of a spherical particle of greater density than the fluid starting from rest derived using the assumptions of Reynolds number Re<<1 and Strouhal number *Sl* known from the literature (Fig. 5. of Lovalenti and Brady [2]) was reproduced.

TEST3: We assumed that the velocity of the fluid at infinity was a constant i.e., $U^{\infty} = U_0$ and we set $\mathrm{Re_F} = 0$. Under these assumptions, we obtained $U_p \rightarrow U_0$ as $t \rightarrow \infty$, which is as expected.

TEST4: We generated a number of outputs using $U_0 = 0$ and $U_0 \neq 0$ and compared the results as $U_0 \rightarrow 0$. We found that the results matched for $U_0 = 0$ and $U_0 \rightarrow 0$, up to an order of 10^{-5}.

TEST5: When we change the initial direction of the motion, namely replacing $\mathrm{Re_F}$ by - $\mathrm{Re_F}$, the phase space plot is reflected about the zero velocity axis. That is, we obtain the reflection of the phase space attractor about the zero velocity axis when the direction of the first motion is reversed. We consider this as an important result which demonstrates the correctness of our results. Our results show a preferred direction in the solution. Since the only physical direction in our problem is the initial direction of the external force, a reversal of that direction should result in a reversal of direction in the solution. We find that this is indeed the case.

TEST6: We observed that there is a shift of position of the attractors when we change the initial condition of Y_p. Changing Y_p at $t = 0$, results only in a shift of attractors as there is only a change in the frame of reference, which does not affect the physics of the problem. This confirms the fact that a change in the initial position only results in a change in coordinate system and not in any physical parameter. This further increases our confidence in our results.

We note here that, $U_p \neq 0$ at $t = 0$, does not make any physical sense, since if we consider $U_p \neq 0$ at $t = 0$, then there must be some particle velocity at negative time too. Moreover, in a quiescent fluid, the particle velocity is due to the application of the external periodic force which is applied only from $t = 0$.

These tests give us considerable confidence in our simuation.

4 Results and Discussions

Typical phase space plots (plots of particle velocity versus particle position) are plotted in Fig. 1. for different values of Re and $\mathrm{Re_F}$. We note that these plots represent a bounded region of phase space and hence the plots represent an attractor in phase space. We observe that here, as $\mathrm{Re_F}$ increases the attractor size also increases, establishing the obvious relation between the amplitude Re_F of the forcing term and the size of the attractors. As the amplitude of the periodic force increases, the particle also oscillates with greater amplitude, covering a greater surface area in the attractor plot. One can observe from the figure that as Re increases, the area bounded by attractors decreases, showing the effect of inertia on the motion of the particle. We can observe that for small Re_F, especially for $Re_F = 0.01$ and different Re, the attractors are different from the other attractors shown in Fig. 1. We note here, that near $Re_F = 0.01$, there exists a bifurcation due to the occurrence of a band of higher harmonics or quasi periodicity in the power spectrum and also the presence of kinks at the two extremes of the phase plots as shown in Fig. 1. At low Re_F, the nonlinearity is comparatively large compared to the external periodic force. Re represents the magnitude of the inertial term, namely a resistance to the change in motion. Hence as Re increases, the resistance to the change in motion also increases diminishing the size of the attractors.

The existence of higher harmonics or quasi periodicity is found in these regimes in the power spectrum. We observe that the power spectrum plots are coherent with that of the Stokes' flow case, showing the effect of periodic force on the motion of the particle dominating over the other parameters. However the phase plots in Fig. 1. follow a small spiral, which is due to the presence of damping term in the expression (5b). One can also note that there is a slight drift in the initial motion of the particle as shown in Fig. 1. From this figure it is evident that there are kinks at the two extremes in the phase space plots which occur around zero magnitudes of the velocity of the particle and hence relatively inertial (nonlinear) effects, since near zero velocity, the rate of change of the velocity is highest. On increasing the resolution of our calculations and hence increasing the sampling frequency, these kinks do not change showing the effects of nonlinear term dominating near $Re_F = 0.01$. Typically, these kinks increase in magnitude at large Re.

When we apply a phase shift of π to the sinusoidal forcing term the attractors shift their direction. That is, when we apply a force in an initially negative direction (the opposite direction) Y_p shows a reflection about the $Y_p = 0$ axis and thus we obtain a reflection of the attractor. The reflection property of these attractors is evident upon reversing the direction of first motion indicating that these features are not a numerical artefact. Figure 2 show the phase plots when the direction of the amplitude of the force is changed and the attractors form a reflection of each other about the axis $Y_p = 0$, as expected. Since the direction of the force represents the direction of initial motion and also there is a fading memory, the particle shows an initial displacement and at large times the periodic motion manifests itself, approximately.

We compared the results of the problem with the Stokes' flow results. We have considered the amplitude ratio as the ratio of the amplitude of the motion of the particle in our problem with the Stokes' flow amplitude. We note that except for $Re_F = 0.01$ and higher Re, the amplitude ratio with respect to Re and Re_F shows a decreasing trend. This might be due to the effect of inertia of the fluid. In the regime of $Re_F = 0.01$, we observe a greater dependence of the amplitude ratio on Re and Re_F. This is reasonable since we see a greater effect of nonlinearity in this region. We also observe that as Re increases the amplitude ratio decreases, showing the effect of inertia is to reduce the amplitude of the motion.

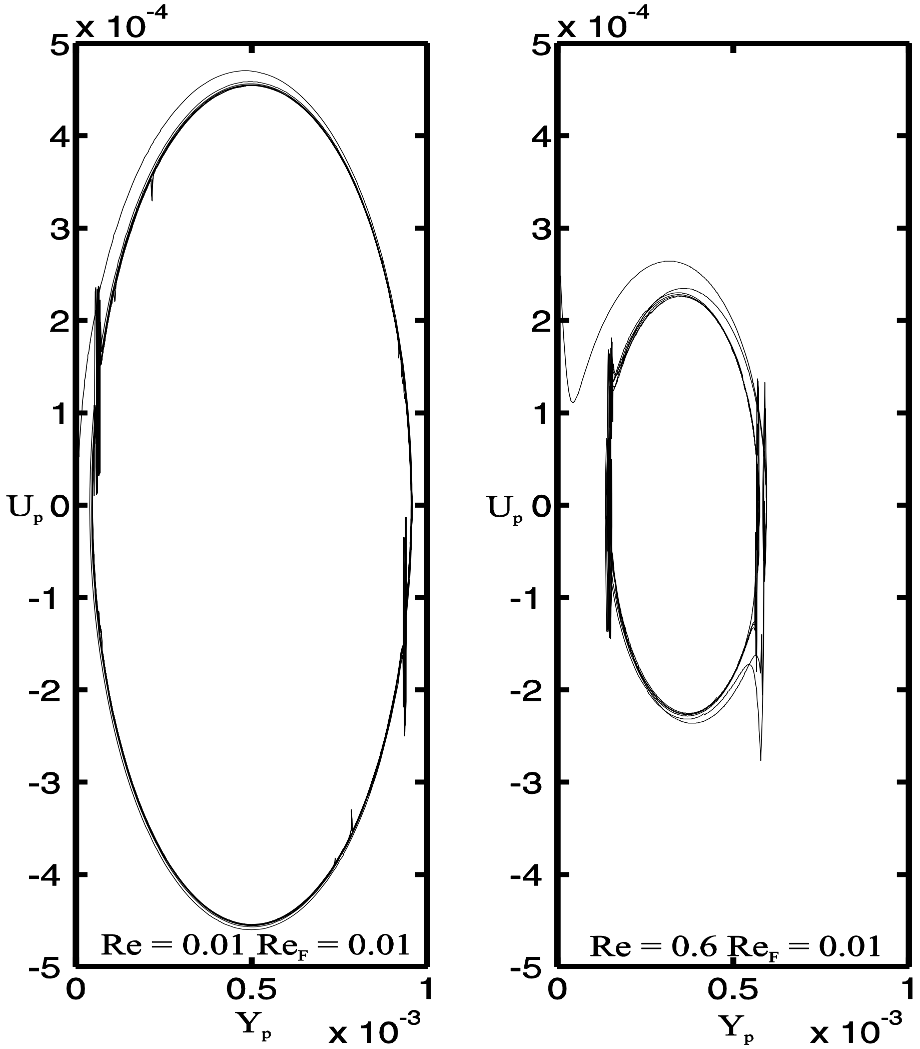


Fig. 1. The phase portrait for various Re and low Re_F. These attractors show kinks at the two extremes for $Re_F = 0.01$, clearly showing the effect of the nonlinearity, namely a bifurcation near Re_F=0.01.

We observed that there is a definite relationship between the mean particle displacement $Y_{p,mean}$ and the amplitude of the external periodic force Re_F. This is as expected since the magnitude of the initial motion is determined by the amplitude of the periodic external force and by the value of Re. As Re increases we note that $Y_{p,mean}$ decreases and as Re_F increases $Y_{p,mean}$ increases.

We obtained the relationship between Re, Re_F and amplitude of the velocity of the particle. We observe that the amplitude of the velocity increases with increase in Re_F and decreases with increase in Re, describing the effect of the periodic force and inertia and effect on the amplitude of the velocity of the particle. Using TISEAN [6], we performed tests for the possibility of chaos in the system and found that there occurs a bifurcation at low Re_F and high Re, there is no chaos in the system.

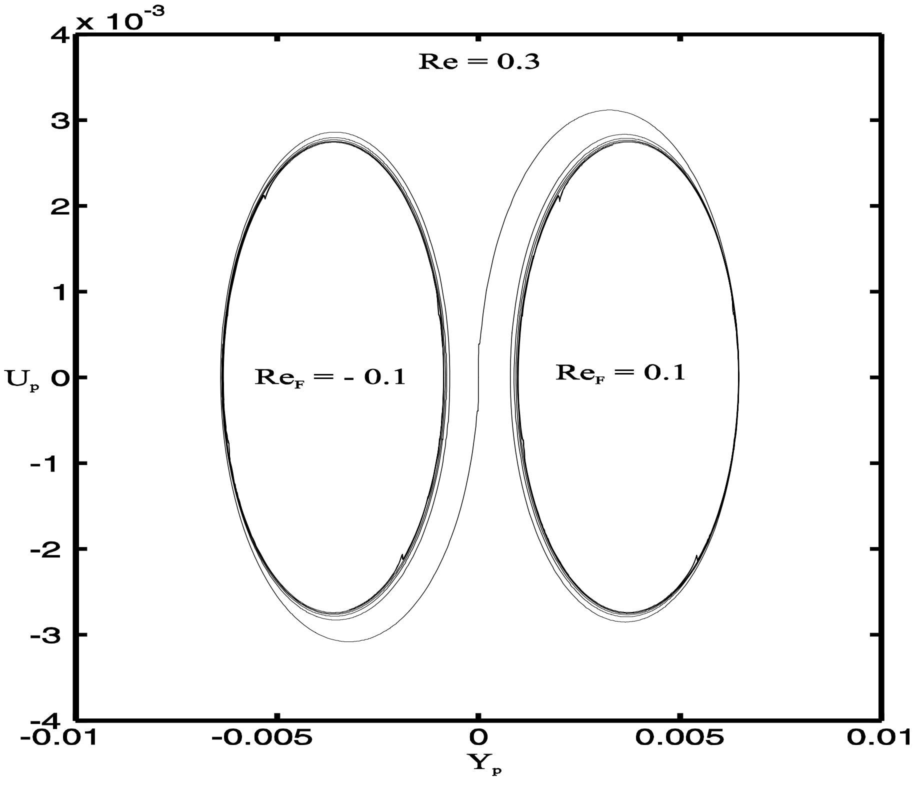


Fig. 2. The phase portrait obtained at Re = 0.3 and $Re_F = \pm 0.1$. The phase portrait shows the reflection property of the solutions of our problem, indicating that there is a physical basis to our results.

5 Conclusion

In this paper, we have attempted to determine the effects of a periodic force on a sphere in a quiescent fluid at low Reynolds numbers. We observe that the particle oscillates around a mean position, due to the periodic force on the particle. There is a net displacement of the mean position of the particle in the direction of first motion. The presence of higher harmonics in the full problem for small Re_F and high Re are noticed and this shows that though there is a nonlinear term in the equation, its effect is small in the parametric regime which we have considered except at low values of Re_F and large values of Re. It is also observed that increasing Re was responsible for an increase in the resistance to the change in particle motion and hence a decrease in attractor area and increasing Re_F leads to an increase in the amplitude of the motion of the particle. Besides, a reflection of the attractor on changing the initial direction of motion is obtained. Our results of the dependence of the mean position of the particle and the amplitude of the velocity of a particle on the problem parameters, such as Re and Re_F may be used to estimate appropriate physical parameters of the system by suitable experiments. We hope that this work will excite further interest in this area.

Acknowledgements

The authors wish to acknowledge Dr. A R Upadhya, Scientist – in – Charge, CSIR-CMMACS, Bangalore-560037, India and Prof. N. Rudraiah, Honorary Professor, UGC – CAS, Department of Mathematics, Bangalore University, Bangalore – 560001, India, for their kind encouragement. The authors also wish to acknowledge the Department of Science and Technology, Govt. of India, New Delhi- 110016, India for financial assistance vide Sanction Letter No. SR/S3/CE/33/2004-SERC Engg. One of the authors, K. Madhukar wishes to thank NAL/C-MMACS for providing the fellowship to do this research work.

References

1. Stokes, G.G.: On the effect of internal friction of fluid on the motion of pendulums. Trans. Camb. Phil. Soc. 9, 8–106 (1851)
2. Lovalenti, P.M., Brady, J.F.: The hydrodynamic force on a rigid particle undergoing arbitrary time – dependent motion at small Reynolds numbers. J. Fluid Mech. 256, 561–605 (1993)
3. Kumar, C.V.A., Ramamohan, T.R.: Controlling chaotic dynamics of periodically forced spheroids in simple shear flow: Results for an example of a potential application. Sadhana 23, 131–149 (1998)
4. Radhakrishnan, K., Asokan, K., Dasan, J., Bhat, C.C., Ramamohan, T.R.: Numerical evidence for the existence of a low dimensional attractor and its implications in the rheology of dilute suspensions of periodically forced slender bodies. Phys. Rev. E. 60, 6602–6609 (1999)
5. Asokan, K., Kumar, C.V.A., Dasan, J., Radhakrishnan, K., Kumar, K.S., Ramamohan, T.R.: Review of chaos in the dynamics and rheology of suspensions of orientable particles in simple shear flow subject to an external periodic force. J. Non-Newtonian Fluid Mech. 129, 128–142 (2005)
6. Hegger, R., Kantz, H., Schreiber, T.: Practical implementation of nonlinear time series methods. CHAOS 9, 413–435 (1999)
7. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical recipes in Fortran. In: The art of scientific computing, 2nd edn. Cambridge University Press, Cambridge (1992)

Bending Virtual Spring-Damper: A Solution to Improve Local Platoon Control

Jean-Michel Contet, Franck Gechter, Pablo Gruer, and Abderrafiaa Koukam

University of Technology of Belfort-Montbeliard (UTBM), Systems and Transportation Laboratory (SET), Belfort, France
{jean-michel.contet,franck.gechter, pablo.gruer,abder.koukam}@utbm.fr
http://set.utbm.fr/

Abstract. This article presents a local control approach to linear vehicle platooning. Linear platoon systems are sets of vehicles that use local or global perception capabilities to form a train configuration, without any hard grip element. Public transportation is beginning to interest in platoon systems as a technological base to conceive new services. The main problem related to platoon system's control corresponds with maintaining inter-vehicle distance. In literature, the platoon's geometry control problem is treated according to two approaches: global or local vehicle control. This paper focuses on a local approach which does not require sophisticated sensors and/or costly road equipment. This local control approach intends to obtain very good global matching to arbitrary trajectories, only from local perception which consists in measuring the vectorial distance between a given vehicle and its predecessor. The behavior of each platoon vehicle is determined from a physics inspired multi agent interaction model based on a virtual spring-damper. Furthermore, stability, platoon safety properties are checked using physics and mathematical proofs. Finally, simulation is used to measure trajectory error and inter-vehicle distance evolution.

Keywords: Platoon, local control, stability proof, simulation.

1 Introduction

Platoon systems can be defined as sets of vehicles that apply local or global perception capabilities to adopt and maintain a train-like or other geometric configuration or formation. Train platoon configurations are also called linear platoons. Control of vehicles' movement within a linear platoon can be decomposed into two main subproblems: longitudinal control and lateral control. Longitudinal control consists in controlling braking and acceleration in order to stabilize the distance between following vehicles. Lateral control consists in determining a vehicle's direction according to the platoon's trajectory. Linear platoon configurations (Cf. Fig. 1) attract considerable attention as an approach to innovative

Fig. 1. Example of linear platoon simulation in front of System and Transportation laboratory (SET)

transportation systems [1]. Their main impact [2], when applied to urban transportation, are: (i) reducing private car affluence (i.e. avoiding traffic jams), (ii) improving security thanks to automatic driving assistance (obstacle detection and avoidance, automatic car parking,...). Platoon based transportation systems are supposed to exhibit a set of convenient properties: adaptability to particular user's service requests, reconfigurability to accommodate a variable number of resources, depending on demand and others.

From the scientific and technological point of view, platoon system's control includes some relevant challenges. The main problem related to platoon systems relates to inter-vehicle distance control. In literature, the control of platoon's global geometry can be divided into two main approach: global or local vehicle control. Within the global approaches each following vehicle is made aware of the leading vehicle trajectory. This solution generally uses some positioning systems such as GPS [9,10]. According to the global control strategy, the leading vehicle broadcasts driving information (steering and speed) to each follower vehicle (i. e., the European *Chauffeur* project [11]). This approach yields very good trajectory following but the global position perception by GPS or other technology implies some road adaptation to avoid tunneling or canyon effects. Moreover, a safe, reliable communication network between vehicle is required. To concluded, the global control approach gives the best results with strong constraints on sensors (high cost), road adaptation and communication reliability between vehicles.

On the other hand, local control consists in computing the vehicle's command references (acceleration and direction) only from vehicle's own perceptions. Most of the lateral or longitudinal control strategies proposed within local approaches based on PID (Proportional, Integral, Derivative) controllers [12,13,14] or other regulation-loop based algorithm [11,15,16,17,23]. Some other works, propose to base on simulated physical phenomena to calculate vehicle's control references. For instance, *Gehrig and Stein* [18] take inspiration from physical particle forces. *Soo-yeong Yi and Kil-to Chong* [19] model immaterial grip by means of an impedance control mechanism. The local approach has been criticized because of their relatively poor trajectory matching and the accordion effect they produce during platoon evolution. Indeed, naive local control approaches increase the trajectory matching error since a follower vehicle tends to anticipate the direction change. On the other hand the local approach, unlike the global one, does not require any sophisticated and highly fallible sensor and can be used in any environment, without any road upgrading.

The paper proposes a local approach which improves the trajectory in any curved trajectory. This approach is based on a virtual spring-damper which bends to avoid curve anticipation. This paper is structured as follows. The Formal (behavior and interaction) Models are presented. Then, stability proofs are presented using mathematical and physical points of view. Finally simulation results are shown. This paper concludes by stressing main contributions and by drawing some perspectives and future works.

2 Formal Models

2.1 Behavioral Model

In the approach presented here, an alternative to centralized control, vehicles are autonomous entities in mutual interaction. Consequently, the intended embodiment for the local platoon control algorithms is a reactive multi-agent system (RMAS). Reactive agents are relatively simple entities that behave based on their own local perceptions [3]. The reactive approach is one of the most interesting due to its robustness, adaptability and simplicity.

The behavior of each vehicle agent is determined from a physics inspired model that minimizes interactions: each vehicle relates only with the preceding one in the platoon. Steady platoon motion over arbitrary trajectories emerges as a global result of individual behaviors. Statechart of Fig. 2 is a high level representation of vehicle agent behavior. The global behavior integrates two

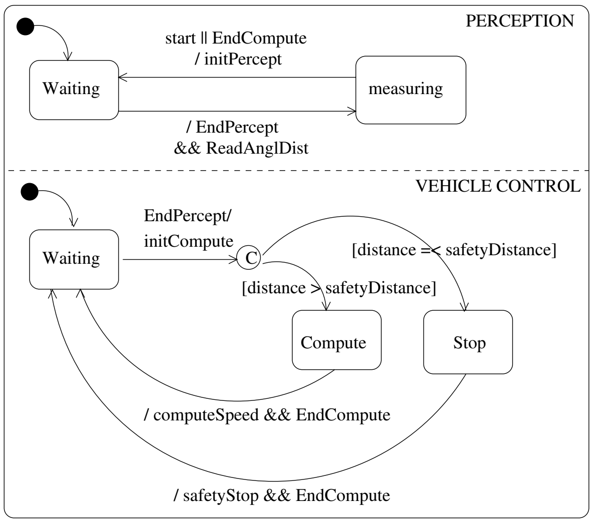


Fig. 2. Inserted vehicle agent behavior

parallels sub behaviors one corresponding to the perception and the other to the action. The vehicle perception behavior correspond to the measure of the inter-vehicle vectorial distance using a sensor such as a laser range finder. When the perception cycle is ended, the vehicle control behavior starts. The A distinction is made between the inter-vehicle distance being under the safety distance or not. The safety distance represents the distance required to perform a safety stop. If the inter-vehicle distance is under the safety distance, function *safetyStop* is called, which triggers a safety stop operation. If the distance is above the safety distance, the function *computeSpeed* is called to compute the new vehicle's speed reference from the physics inspired interaction model.

2.2 Interaction Model

Physics inspired interaction model is used to specify the response of an agent, as determined by its perceptions. The model proposed in this paper is based on three virtual forces: spring force $\boldsymbol{F}_s$, damping force $\boldsymbol{F}_d$ and surface friction force $\boldsymbol{F}_f$, as in [5,6], augmented by torsion force $\boldsymbol{F}_l$, introduced to improve the interaction model in curved trajectories, by adding a flexion force to the impedance control model. The virtual link forces are calculated from a classical spring damper model with stiffness k, damping h and spring's un stretched length l_0. Each vehicle i is represented by its position $\boldsymbol{X}_i = [x_i, y_i]$. The mass of the vehicle is denoted as m (we assume that each vehicle has the same mass). The distance between vehicles is $d = \|\boldsymbol{X}_{n+1} - \boldsymbol{X}_n\|$.

Longitudinal and lateral control references can be determined from just these three forces ($\boldsymbol{F}_s$, $\boldsymbol{F}_d$, $\boldsymbol{F}_f$), by calculating a new acceleration value. However, simulations and experimentations [6] have exhibited an increase in trajectory error from one vehicle to its follower. This error is due to limitations of the linear impedance control model (Cf. Fig. 3 left). Figure 3 shows the acceleration anticipation and its consequence, trajectory error. To avoid this problem, we propose to add a force, due to the flexing of the impedance control. This force counters the acceleration anticipation. This force is deduced from a virtual moment, result of a flexing spring $\boldsymbol{Moment} = k_m * \theta * \boldsymbol{z}$ with k_m the torsion spring parameter and θ the spring angle of flection. This force is computed as

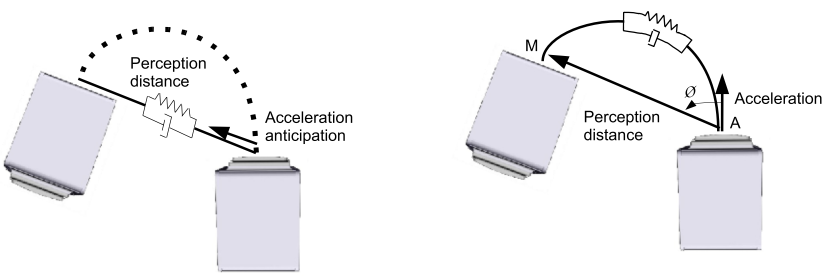


Fig. 3. Acceleration problem and issue

the vector product of the distance between the moment application point and the preceding vehicle position (C.f Fig. 3 right).

Those forces allow to compute the vehicle's acceleration, by using Newton's law of motion determined relatively to the preceding vehicle reference:

Spring force	$\boldsymbol{F}_s = -k(\|\boldsymbol{X}_{n+1} - \boldsymbol{X}_n\| - l_0)\boldsymbol{u_{n+1\ n}}$
Damping force	$\boldsymbol{F}_d = -h(\dot{\boldsymbol{X}}_{n+1} - \dot{\boldsymbol{X}}_n)$
Friction force of the surface	$\boldsymbol{F}_f = -\lambda\dot{\boldsymbol{X}}_n$
Torsion force	$\boldsymbol{F}_{torsion} = \boldsymbol{MA} \wedge \boldsymbol{Moment}$

Acceleration value:

$$m{*}\boldsymbol{\gamma} = -k(\|\boldsymbol{X}_{n+1}-\boldsymbol{X}_n\|-l_0)\boldsymbol{u_{n+1\ n}}-h(\dot{\boldsymbol{X}}_{n+1}-\dot{\boldsymbol{X}}_n)+-\lambda\dot{\boldsymbol{X}}_n+\boldsymbol{MA}\wedge\boldsymbol{Moment} \tag{1}$$

By discrete integration, we can then determine speed and vehicle state (position and orientation). Then the command reference can be computed. In our case, it consists in vehicle direction and speed [1]. Model parameters are determined by considering the vehicle's characteristics and constraints.

3 Inter-Vehicle Distance Stability Analysis

Analysis of the interaction model by applying classical laws allows to prove that the interaction model satisfies a set of platoon stability properties. The next section presents two kinds of proofs: one applying a direct approach and the other using an energetic point of view.

3.1 Platoon Stability Based on Transfer Function

Platoon stability is treated following the "string stability" problem which has been studied in 1977 by Caudill and Garrard [8]. A platoon is said to be stable if, under no other excitations, the error magnitude decreases as it propagates along the vehicle stream. Here, we proposed to prove the string stability relatively to longitudinal control. Formally speaking, if the transfer function of the system composed of two successive vehicles exhibits a magnitude less or equal to 1, string stability is obtained [21,22]. In order to verify this property, the control law of vehicles i and $i-1$ must be expressed with $\theta = 0$ (Longitudinal control): let us suppose A equal to $\frac{\boldsymbol{X}(s)_i}{\boldsymbol{X}(s)_{i-1}}$. From laws of motion applied to vehicles i and $i-1$, we obtain

$$\begin{cases} m * \ddot{\boldsymbol{X}}_i = -k\boldsymbol{X}_i - h\dot{\boldsymbol{X}}_i \\ m * \ddot{\boldsymbol{X}}_{i-1} = -k\boldsymbol{X}_{i-1} - h\dot{\boldsymbol{X}}_{i-1} \end{cases} \tag{2}$$

From these equations A can be deduced: $A = \frac{h*s+k}{s^2+h*s+k}$.
String stability is guaranteed only if: $|A(j\omega)| \leq 1$. This condition is verified with $k \leq \frac{m*\omega^2}{2}$. This study proof that our system is stable to any frequency. The error propagation will be attenuated through the platoon.

[1] The choice of a command law takes into account the characteristics of the test vehicle used in our laboratory.

3.2 Inter-Vehicle Distance Stability Proof Based on Energy

We intend to check if the stability[2] of the system is kept in run time. To this end we apply Lyapunov stability of motion conditions [7,20]. The system energy results from the addition of kinetic and potential energies. We take vehicle i as reference frame to express the energy.

$$\begin{aligned} E &= E_{kinetics} + E_{potential} = E_{kinetics} + E_{pot}(F_s) + E_{pot}(F_{torsion}) \\ E &= \tfrac{1}{2} * m * (\dot{\boldsymbol{X}}.\dot{\boldsymbol{X}}) + \tfrac{1}{2} * k * (\boldsymbol{X}.\boldsymbol{X}) + \int_X \int_\theta \boldsymbol{F}.dY d\theta \end{aligned} \tag{3}$$

By applying Lyapunov stability of motion conditions [7,20], we obtain

$$\begin{bmatrix} \frac{\partial E(x,y,\theta)}{\partial x} \\ \frac{\partial E(x,y,\theta)}{\partial y} \end{bmatrix} = \begin{bmatrix} x_2^2(k_m sin(\theta) - h) \\ y_2^2(k_m cos(\theta) - h) \end{bmatrix} \tag{4}$$

The derivative of Energy is negative if the angular $h * k_m$ is greater than 1 since $\theta \in [-\pi/2, \pi/2]$. Applying Lyapunov stability of motion conditions as in [6,4], shows that the system is stable when time tends to infinity. Thus, the distance between vehicles tends to the un stretched spring length when the platoon moves without environmental influence, if the condition on rotation angle ($\theta \in [-\pi/2, \pi/2]$) is verified.

4 Experimentation Results

The physics inspired interaction model previously defined was the base to the specification of the local platooning control algorithms. Based on those algorithms, simulation experiments were designed and performed to check a set of safety platoon conditions. The simulations have been made on a simulator able to take into account vehicle dynamics properties. The simulator has 3D graphic display capabilities developed with OpenSceneGraph[3], in connection with a dynamics engine[4] used to model the physical aspects (realistic model of a car moving in a material environment.) The vehicle dynamics have been modeled as a physical box with 4 shock absorber wheels. Vehicle models integrate the dynamic properties of a real vehicle. Simulations have been realized in order to visualize inter-vehicle distance and vehicle position evolution within a platoon (C.f Fig. 4).

In order to perceive the environment and the preceding vehicle, the model of a laser range finder has been included in the follower vehicles. The laser measurement system is based on a time-of-flight measurement principle. Simulations have been performed to verify the platoon evolution during startup and lateral displacement situations.

[2] platoon stability is represented by the stabilization of the distance between each follower vehicle.

[3] http://www.openscenegraph.org/projects/osg

[4] http://www.ode.org/

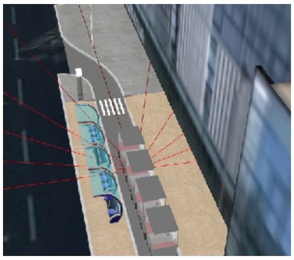


Fig. 4. 3 Dimensions simulator: platoon of 5 vehicles into a station (rays correspond to the second vehicle laser range finder)

4.1 Platoon Startup

The simulation starts with 4 platoon vehicles at 1.6 meter of inter-vehicle distance.

The first vehicle is controlled by the user. Each follower vehicle follows the preceding one, by executing its own perception-based behavior. Figure 5 shows the inter-vehicle distance during the platoon evolution. The inter-vehicle distance value is initialized to 1.6 meter with a vehicle speed of 30 km/h then a safety stop is performed. All inter-vehicle distances decrease to 0.5 meter. This shows that there are no collisions but only some little *oscillations in simulations.*

4.2 Platoon Lateral Displacement

Platoon lateral displacement simulation (station entering and exiting) have been realized to check that each vehicle follows correctly its predecessor.

Figures 6 illustrates platoon evolution during station exiting, by tracing both the leader and follower trajectories. This figure exhibits a little increase in

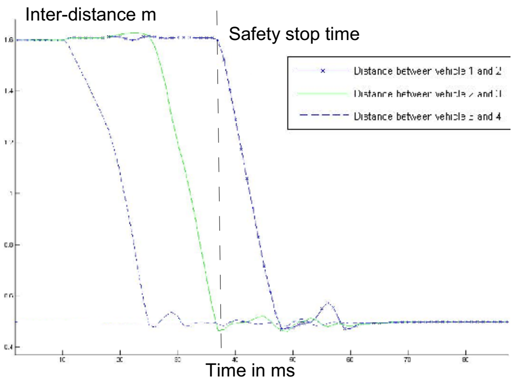


Fig. 5. 3D simulator: inter-vehicle distance evolution

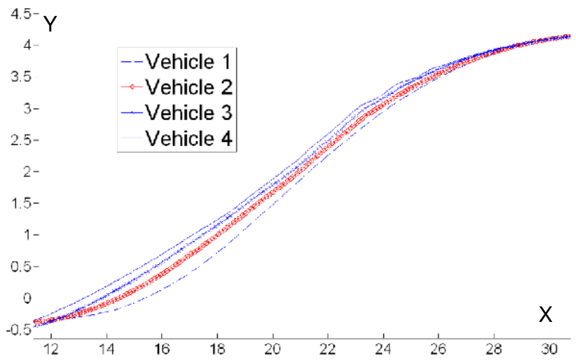
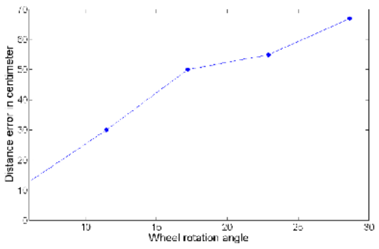


Fig. 6. 3D simulator: Trajectory error during the station exit



Wheel rotation (degree)	Curve radius	medium error	maximum error
5.73	18 m	12 cm	14 cm
11.46	9 m	30 cm	40 cm
17.2	6 m	50 cm	65 cm
22.9	4.5 m	55 cm	67 cm
28.65	3.6 m	67 cm	75 cm

Fig. 7. 3D simulator: trajectory error in curve

trajectory error from one vehicle to its follower. The maximum of this error does not exceed the width of a wheel (C.f Fig. 6 left).

4.3 Relationship between the Curve and the Trajectory Error

The simulation presented next was intended to show the relationship between the bend radius and the trajectory error as presented previously.

This simulation corresponds to the vehicle starting with a given wheel angle. Then, the simulator computes the evolution of the distance error. Figures 7 and table illustrates the error in the curve. This error increases with the magnitude of the wheel angle.

5 Conclusion

The aim of this article was to present a better local platoon control based on the bending of a virtual spring damper. Platoon control, as presented in this paper, bases only on local perception capabilities. Each vehicle behavior is deduced from a physics inspired interaction model an embodied in a reactive agent architecture. The use of physics inspired forces enables an easier tuning of the interaction model parameters and the adaptation to any kind of vehicle. Besides, the physic model has been used to prove platoon stability, by using classic physical proof method: energy analysis. Analogously, another stability proof have been realized following a transfer-function approach. To assert the transition from abstract to concrete, simulations have been implemented to show the applicability of the theoretical model. Some simulations have been made on a 3D simulator which integrates vehicle dynamic properties (maximal speed and acceleration, ...). These experimentations exhibit a little curved trajectory error during the platoon evolution and indicate that the presented approach improves platoon quality. We are now working on the application of this model on real vehicles. Moreover, we are also advancing into further research on the use of Formal Models in order to prove some application properties and to ensure a zero default embedded software for real vehicles.

References

1. Hedrick, J.K., Tomizuka, M., Varaiya, P.: Control issues in automated highway systems. IEEE Control Systems Magazine, 21–32 (1994)
2. Debojyoti, M., Asis, M.: Pollution control by reduction of drag on cars and buses through platooning. Int. J. of Environment and Pollution 30(1), 90–96 (2007)
3. Ferber, J.: Multi-Agent System: An Introduction to Distributed Artificial Intelligence. Addison Wesley Longman, Harlow (1999)
4. Gaud, N., Gechter, F., Galland, S., Koukam, A.: Holonic multiagent multilevel simulation: Application to real-time pedestrians simulation in urban environment. In: Twentieth International Joint Conference on Artificial Intelligence, IJCAI 2007, pp. 1275–1280 (2007)
5. Contet, J.M., Gechter, F., Gruer, P., Koukam, A.: Physics inspired multiagent model for vehicle platooning. In: International Conference on Autonomous Agents and Multiagent Systems AAMAS (2007)
6. Contet, J.M., Gechter, F., Gruer, P., Koukam, A.: Multiagent System Model for Vehicle Platooning with Merge and Split Capabilities. In: Third International Conference on Autonomous Robots and Agents, pp. 41–46 (2006)
7. Taylor, F., Lyapunov, A.M.: The general problem of the stability of the motion (1992)
8. Garrard, W.L., Caudill, R.J.: Dynamic Behavior of Strings of Automated Transit Vehicles. SAE Transactions, 1365–1378 (1977)
9. Martinet, P., Thuilot, B., Bom, J.: Autonomous Navigation and Platooning using a Sensory Memory. In: International IEEE Conference on Intelligent Robots and Systems, IROS 2006, Beijing, China (2006)
10. Myung, J.W., Jae Weon, C.: A relative navigation system for vehicle platooning. In: SICE 2001, Proceedings of the 40th SICE Annual Conference, pp. 28–31 (2001)
11. Fritz, H.: Longitudinal and lateral control of heavy duty trucks for automated vehicle following in mixed traffic: Experimental results from the CHAUFFEUR project. In: IEEE Conference on Control Applications - Proceedings, vol. 2, pp. 1348–1352 (1999)
12. Ioannou, P., Xu, Z.: Throttle and brake control systems for automatic vehicle following. IVHS Journal, 345 (1994)
13. Moskwa, J.J., Hedrick, K.J.: Nonlinear algorithms for automotive engine control. IEEE Control Systems Magazine 10, 88–93 (1990)

14. Daviet, P., Parent, M.: Longitudinal and lateral servoing of vehicles in a platoon. In: Proceedings of IEEE Intelligent Vehicles Symposium, pp. 41–46 (1996)
15. Sheikholeslam, S., Desoer, C.A.: Longitudinal control of a platoon of vehicles with no communication of lead vehicle information: A system level study. IEEE Transactions on Vehicular Technology 42, 546–554 (1993)
16. Lee, H., Tomizuka, M.: Adaptive vehicle traction force control for intelligent vehicle highway systems (IVHSs). IEEE Transactions on Industrial Electronics 50, 37–47 (2003)
17. Kehtarnavaz, N., Griswold, N.C., Lee, J.S.: Visual control of an autonomous vehicle (BART)–The vehicle-following problem. IEEE Transactions on Vehicular Technology 40, 654–662 (1991)
18. Gehrig, S.K., Stein, F.J.: Elastic bands to enhance vehicle following. In: IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC, pp. 597–602 (2001)
19. Yi, S.-Y., Chong, K.-T.: Impedance control for a vehicle platoon system, Mechatronics (UK), vol. 15, p. 627 (2005)
20. Leipholz, H.: An introduction to the stability of Dynamic systems and rigid bodies (1987)
21. Swaroop, D., Hedrick, J.K.: String stability of interconnected systems. IEEE Transactions on Automatic Control 41, 349–357 (1996)
22. Liang, C., Peng, H.: Optimal Adaptive Cruise Control with Guaranteed String Stability (1998)
23. Klancar, G., Matko, D., Blazic, S.: Wheeled Mobile Robots Control in a Linear Platoon. Journal of Intelligent and Robotic System (2008)

Parallel Algorithms for Dandelion-Like Codes*

Saverio Caminiti and Rossella Petreschi

Computer Science Department, *Sapienza* University of Rome
Via Salaria, 113 - I00198 Rome, Italy
{caminiti,petreschi}@di.uniroma1.it

Abstract. We consider the class of Dandelion-like codes, i.e., a class of bijective codes for coding labeled trees by means of strings of node labels. In the literature it is possible to find optimal sequential algorithms for codes belonging to this class, but, for the best of our knowledge, no parallel algorithm is reported. In this paper we present the first encoding and decoding parallel algorithms for Dandelion-like codes. Namely, we design a unique encoding algorithm and a unique decoding algorithm that, properly parametrized, can be used for all Dandelion-like codes. These algorithms are optimal in the sequential setting. The encoding algorithm implementation on an EREW PRAM is optimal, while the efficient implementation of the decoding algorithm requires concurrent reading.

1 Introduction

Trees are one of the most studied class of graphs in Computer Science; they are used in a large variety of domains, including computer networks, computational biology, databases, pattern recognition, and web mining. In almost all applications, tree nodes and edges are associated with labels, weights, or costs. Examples range from XML data to tree-based dictionaries (heaps, AVL, RB-trees), from phylogenetic trees to spanning trees of communication networks, from indexes to tries (used in compression algorithms). Many are the usual representations of tree data structures: adjacency matrices, adjacency lists, parent vectors, and balanced parentheses are just a few examples. An interesting alternative is based on coding labeled trees by means of strings of node labels.

String-based codes for labeled trees have many practical applications. For example, they are used in fault dictionary storage [1], distributed spanning tree maintenance [2], generation of random trees [3], Genetic Algorithms [4]. In this paper we restrict our attention to bijective string-based codes in which the length of the string must be equal to $n-2$ [5] (n is the number of nodes of the encoded tree). The first bijective string-based coding for trees is due to Prüfer [6]. since then, many other bijective codes have been introduced [7,8,9,10,11,12,13,14].

Concerning algorithmic aspects of these codes, there is a wide literature presenting optimal sequential algorithms for encoding and decoding all of them. In particular we refer to [15] for the class of Prüfer-like codes (the first 4 codes in Table 1)

* Work partially supported by *Sapienza* University of Rome under the project "Strutture Dati e Tecniche Algoritmiche Evolute per Modelli di Calcolo Innovativi".

Table 1. Costs of known algorithms for bijective string-based codes. Parallel costs are expressed as the number of processors $\times$ maximum execution time.

	Sequential		Parallel	
	Encoding	Decoding	Encoding	Decoding
Prüfer [6]	$O(n)$	$O(n)$	$O(n)$	$O(n\sqrt{\log n})$
2nd Neville [13]	$O(n)$	$O(n)$	$O(n\sqrt{\log n})$	$O(n\sqrt{\log n})$
3rd Neville [13]	$O(n)$	$O(n)$	$O(n)$	$O(n\sqrt{\log n})$
Stack-Queue [9]	$O(n)$	$O(n)$	$O(n\sqrt{\log n})$	$O(n\sqrt{\log n})$
θ_n [10]	$O(n)$	$O(n)$	unknown	unknown
Happy [14]	$O(n)$	$O(n)$	unknown	unknown
Dandelion [14]	$O(n)$	$O(n)$	unknown	unknown
MHappy [7]	$O(n)$	$O(n)$	unknown	unknown
Blob [11,14]	$O(n)$	$O(n)$	unknown	unknown
Chen [8]	$O(n)$	$O(n)$	unknown	unknown

and to [7] for the other codes. A survey of optimal sequential algorithms can be found in [16]. Also parallel algorithms have been studied [7,17,18,19], but results are known only for Prüfer-like codes. More results related to sequential and parallel algorithms for bijective codes can be found in [20] and are summarized in Table 1.

In this paper we make a further step in understanding the feasibility of encoding and decoding in a parallel setting. We focus on the class of Dandelion-like codes introduced in [21] and we present efficient parallel algorithms for encoding and decoding all these codes. This class contains the second block of codes (rows 5 to 8) in Table 1. Dandelion-like codes are especially useful in Genetic Algorithms (GA) since experimental analysis show that Prüfer-like codes perform poorly (with respect to GA requirements) [22] while Dandelion-like codes achieve best results [21]. The techniques we use for Dandelion-like codes can be also exploited to obtain efficient parallel algorithms for encoding and decoding the Blob code (rows 5 in Table 1).

The paper is organized as follows: after a few preliminary definitions, in Sec.3 we recall the Dandelion code, together with examples. In Sec.4 we recall the class of Dandelion-like codes and introduce an encoding and a decoding algorithm that, properly parametrized, can be used for all codes in the class. In a sequential setting these two algorithms run in linear time and are therefore optimal. In Sec.6 we discuss how to parallelize our algorithms for the PRAM model. For the encoding algorithm we obtain optimal linear cost implementation on an EREW PRAM. The decoding algorithm requires $O(n \log n)$ cost on a CREW PRAM.

2 Preliminary Definitions

In this section, we introduce some definitions that will be useful in the rest of this paper. As usual, the notation $[a, b]$ represents the integer interval from a to b, both included.

Definition 1. *A* labeled n-tree *is an unrooted tree with* n *nodes, each with a distinct label selected in the set* $[0, n-1]$.

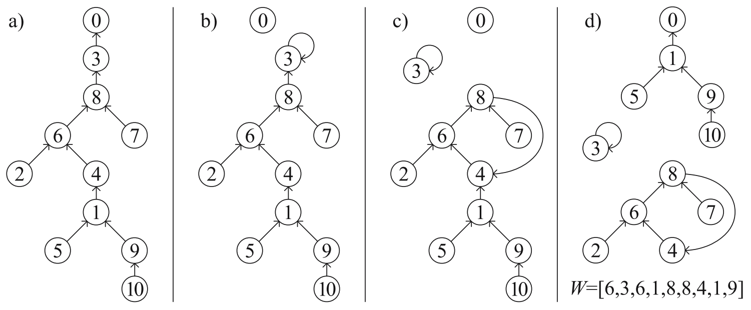


Fig. 1. Example of Dandelion encoding algorithm execution

Since in this paper we only deal with labeled trees, we will refer to them simply as trees. In the following all trees will be regarded as rooted in the fixed node 0 and its edges will be oriented upward from a node to its parent; an example is depicted in Fig.1a.

Definition 2. *Given a function g from the set $[0, n]$ to the set $[0, n]$, the* functional digraph $G = (V, E)$ *associated with g is a directed graph with $V = \{0, \ldots, n\}$ and $E = \{(v, g(v))$ for every $v \in V\}$.*

For this class of graphs the following lemma holds:

Lemma 1. *A digraph $G = (V, E)$ is a functional digraph if and only if the out-degree of each node is equal to 1.*

Functional digraphs are easily generalizable for representing functions which are undefined in some values: if $g(x)$ is not defined, the node x in G does not have outgoing edges.

As an example consider a rooted tree T and let $p[v]$ be the parent of v for each v in T. T is the functional digraph associated with the function p.

In the following, when no confusion arise, we will consider vectors as functions and vice versa. The notation $u \rightsquigarrow v$ identifies the directed path from node u to node v; $u \rightsquigarrow u$ is the degenerate path of length 0. Loops will be considered as cycles of length 1.

3 The Dandelion Code

We now recall the Dandelion code (originally introduced in [14]) as reinterpreted in [7] in which a tree T is transformed into a functional digraph G_g. Initially $G_g = T$, thus the function g is equivalent to the parent vector of T. The Dandelion code choose node 1 to play a special role and rearrange all nodes in $1 \rightsquigarrow 0$ into cycles; the resulting digraph will correspond to a function g such that $g[0]$ is undefined and $g[1] = 0$. Let us detail the rearrangement of nodes in $1 \rightsquigarrow 0$:

Let $1 = v_1, v_2, \ldots, v_l = 0$ be all nodes in $1 \rightsquigarrow 0$ and let $m_i = \max\{v_i, \ldots, v_m\}$. Among them we choose all nodes such that $m_i = v_i$ (excluding v_l); these nodes identify a subsequence $f_1, f_2, \ldots, f_k$ of sequence $v_1, v_2, \ldots, v_{l-1}$. We assign $f_0 = 1$. As an example consider the tree of 11 nodes, labeled from 0 to 10 in Fig.1a in which the path $1 \rightsquigarrow 0$ is (1, 4, 6, 8, 3, 0). We obtain $f_0 = 1$, $f_1 = 8$, $f_2 = 3$. The algorithm proceeds by set $g[f_i] = g[f_{i-1}]$ for each i from k down to 1. During this process all nodes in the original path between 1 and 0 are partitioned into several cycles (or loops). At the end the path $1 \rightsquigarrow 0$ is reduced to a single edge $(1, 0)$ and the resulting codeword is $g[2], g[3], \ldots, g[n-1]$. In our example, since $f_0 = 1$, $f_1 = 8$, and $f_2 = 3$, the algorithm set $g[3] \leftarrow g[8] = 3$, introducing a loop, and $g[8] \leftarrow g[1] = 4$, introducing a cycle. Fig.1d shows the resulting digraph: the associated codeword is $W = [6, 3, 6, 1, 8, 8, 4, 1, 9]$.

The procedure can be easily inverted to obtain the tree T corresponding to any given codeword $W = w_1, w_2, \ldots, w_{n-2}$. Initially reconstruct the functional digraph corresponding to g defined as: $g[0]$ is undefined, $g[1] = 0$, and $g[i] = w_{i-1}$. Then, identify all cycles $C_1, C_2, \ldots, C_k$ in G_g. For each cycle C_i, let us call *characteristic* its maximum node $f_i = \max\{v \in C_i\}$. W.l.o.g. assume that characteristic nodes are numbered such that $f_1 > f_2 > \ldots > f_k$.

The reconstruction of the original path $1 \rightsquigarrow 0$ is obtained reinserting all nodes of each cycle in between 1 and 0. Cycles are selected in descending order with respect to their characteristic values and, for each i from 0 to $k-1$, we set $g[f_i] = g[f_{i+1}]$ (where $f_0 = 1$). At the end we set $g[f_k] = 0$: G_g is now equal to the tree T corresponding to the given codeword W.

As an example of decoding consider the codeword $W = [6, 3, 6, 1, 8, 8, 4, 1, 9]$. It is easy to see that, according with the reconstruction described above, we initially obtain the functional digraph represented in Fig.1d. This graph has two cycles: one is a loop on node 3, and one is induced by nodes 6, 8, and 4. Thus we obtain $f_0 = 1$, $f_1 = 8$, $f_2 = 3$. Then we set $g[1] \leftarrow g[8] = 4$, $g[8] \leftarrow g[3] = 3$, and $g[3] \leftarrow 0$. Now G_g is exactly the tree represented in Fig.1a.

4 Dandelion-Like Codes

In this section we report the class of Dandelion-like codes introduced in [21]. We describe them in terms of functional digraphs in the style of [7] and we explicitly present one optimal encoding and one optimal decoding parametrized algorithms that can be used for all these codes.

Looking at the Dandelion code presented in Sec.3 it is easy to see that several details can be changed to originate different codes:

1. use minimum instead of maximum to compute f_i nodes among those in the path $1 \rightsquigarrow 0$ (the characteristic nodes in the decoding procedure);
2. search downward in the path from f_i to 1 instead of searching upward in the path from f_i to 0;
3. invert the orientation of all edges in cycles.

A summary of the 2^3 bijective codes obtained by all possible combinations of the three changes is reported in the following table – it is to notice that in [21] the 8 bijective codes reported were selected among 16 codes generated considering 4 possible changes of the decoding phase. Codes are numbered according with [21].

Code	max/min	up/down	edge orientation	Code name
C_1	max	up	preserve	Dandelion [14]
C_2	max	down	invert	Happy [14]
C_3	max	down	preserve	MHappy [7]
C_4	max	up	invert	
C_5	min	up	preserve	θ_n bijection [10]
C_6	min	down	invert	
C_7	min	down	preserve	
C_8	min	up	invert	

Encoding Algorithm. In this section we present an algorithm able to encode all the 8 Dandelion-like codes. In Program 1 we report only the fragment of code that transforms a tree into a functional digraph: obtaining the codeword from the functional digraph and vice versa is straightforward (see Sec.3). The Dandelion-Like Encoding Algorithm has three parameters:

1. $\mu \in \{min, max\}$ specifies whether to search for minimum or maximum values;
2. $\uparrow\!\downarrow \in \{up, down\}$ establishes if the μ values should be searched upward or downward;
3. invertEdges is a boolean value that discriminates whether the orientation of cycle edges should be inverted or not.

The value $\mu_{\uparrow\downarrow}(v)$ represents, for each v in the path $1 \rightsquigarrow 0$, the maximum/ minimum value above/below node v (including v itself). Thus, depending on the parameters μ and $\uparrow\!\downarrow$, the function $\mu_{\uparrow\downarrow}(v)$ can be one of the followings:

$$\max_{\text{up}}(v) = \max\{w \in v \rightsquigarrow 0\} \qquad \max_{\text{down}}(v) = \max\{w \in 1 \rightsquigarrow v\}$$
$$\min_{\text{up}}(v) = \min\{w \in v \rightsquigarrow 0\} \qquad \min_{\text{down}}(v) = \min\{w \in 1 \rightsquigarrow v\}$$

Let us now analyze the algorithm. All values $\mu_{\uparrow\downarrow}(v)$ can be computed with simple forward/backward scan of the path $1 \rightsquigarrow 0$; at the same time all f_i are identified. This requires $O(n)$ time. In line 4, if $\uparrow\!\downarrow$ = down, the highest node before f_{i+1} is identified. Indeed, in this case, f_i should form a cycle with all node above it and below f_{i+1}. This operation can be performed in linear time traversing the path $1 \rightsquigarrow 0$ once again. The same time bound holds for lines 5–6. Line 7 can efficiently be implemented in the following way: each node v in the path $1 \rightsquigarrow 0$ sets itself as the new parent of its current parent (i.e., $p'[p[v]] = v$) and all values in p are updated according with the values in p'. The overall time complexity of the Dandelion-Like Encoding Algorithm is linear.

Consider, as an example, the encoding of the tree shown in Fig.1a with all possible codes. The various codes identify the following nodes in line 2:

$$C_1, C_4 : f_1 = 8, f_2 = 3 \qquad C_2, C_3 : f_1 = 4, f_2 = 6, f_3 = 8$$
$$C_5, C_8 : f_1 = 3 \qquad C_6, C_7 : f_1 = 4, f_2 = 3$$

thus introducing the following cycles in the functional digraph G_p:

$$C_1, C_4 : (4, 6, 8), (3) \qquad C_2, C_3 : (4), (6), (8, 3)$$
$$C_5, C_8 : (4, 6, 8, 3) \qquad C_6, C_7 : (4, 6, 8), (3)$$

The resulting codewords are: C_1 = [6, 3, 6, 1, 8, 8, 4, 1, 9], C_2 = [6, 8, 4, 1, 6, 8, 3, 1, 9], C_3 = [6, 8, 4, 1, 6, 8, 3, 1, 9], C_4 = [6, 3, 8, 1, 4, 8, 6, 1, 9],

Program 1. Dandelion-Like Encoding Algorithm

Parameters: μ, $\upharpoonleft\downharpoonleft$, INVERTEDGES
Input: a tree T represented by its parent vector p
Output: a functional digraph G_p such that $p[0] = undef$ and $p[1] = 0$

```
1. Compute $\mu_{\upharpoonleft\downharpoonleft}(v)$ for each $v$ in $1 \rightsquigarrow 0$ excluding 1 and 0
2. Identify all nodes $f_1, f_2, \ldots, f_k$ such that $\mu_{\upharpoonleft\downharpoonleft}(f_i) = f_i$
3. $f_0 = 1$; $f_{k+1} = 0$
4. if $\upharpoonleft\downharpoonleft = \text{down}$ then $f_i = \{v \in 1 \rightsquigarrow 0 : p[v] = f_{i+1}\}$  $\forall\, 1 \le i \le k$
5. for $i = k$ down to 1 do $p[f_i] = p[f_{i-1}]$
6. $p[1] = 0$
7. if INVERTEDGES then invert all edges in cycles
```

Program 2. Dandelion-Like Decoding Algorithm

Parameters: μ, $\upharpoonleft\downharpoonleft$, INVERTEDGES
Input: a functional digraph G_p such that $p[0] = undef$ and $p[1] = 0$
Output: a tree T represented by its parent vector p

```
1. Find all cycles $C_i$ and their characteristic nodes $f_i$ according to $\mu$
2. if ($\mu = \max$ and $\upharpoonleft\downharpoonleft = \text{up}$) or ($\mu = \min$ and $\upharpoonleft\downharpoonleft = \text{down}$) then
3.   Sort $\{f_i\}$ in decreasing order
4. else Sort $\{f_i\}$ in increasing order
5. if INVERTEDGES then invert all edges in cycles
6. $f_0 = 1$; $f_{k+1} = 0$
7. if $\upharpoonleft\downharpoonleft = \text{down}$ then $f_i = \{v \in C_i : p[v] = f_i\}$  $\forall\, 1 \le i \le k$
8. for $i = 0$ to $k-1$ do $p[f_i] = p[f_{i+1}]$
9. $p[f_k] = 0$
```

C_5 = [6, 4, 6, 1, 8, 8, 3, 1, 9], C_6 = [6, 3, 8, 1, 4, 8, 6, 1, 9], C_7 = [6, 3, 6, 1, 8, 8, 4, 1, 9], and C_8 = [6, 8, 3, 1, 4, 8, 6, 1, 9].

Notice that the 8 codes are all different from each other, even thought, on this small example, different codes produce the same codeword.

Decoding Algorithm. We now describe how to invert the transformation: given a functional digraph we identify its cycles and compute, for each cycle C_i the characteristic node f_i according with the function specified by the parameter μ. Then all cycles are broken and their nodes are placed in between 1 and 0 in such a way that the original path $1 \rightsquigarrow 0$ of the tree is reconstructed. If INVERTEDGES is true then, before breaking cycles, the edge orientation should be reestablished.

Let us now describe how to reconstruct the correct order among the characteristic nodes. If $\mu = \max$ and $\upharpoonleft\downharpoonleft = \text{up}$ then greater f_i must be below any other characteristic node, thus the f_i values must be ordered in decreasing order: $f_1 > f_2 > \ldots > f_k$. On the other hand, if $\upharpoonleft\downharpoonleft = \text{down}$ then the greater f_i must be placed above any other characteristic node, thus implying an increasing order: $f_1 < f_2 < \ldots < f_k$. If $\mu = \min$ the orders are reversed. So, the path $1 \rightsquigarrow 0$ have to be rebuilt according with the ordering of the f_i: $1 \rightsquigarrow f_1 \rightsquigarrow \ldots \rightsquigarrow f_k \rightsquigarrow 0$.

Finally notice that, if $\upharpoonleft\downharpoonleft = \text{up}$ then the characteristic node should be above all nodes of its cycle, otherwise it should be below them. This is all we need to correctly rebuild the path $1 \rightsquigarrow 0$ of the original encoded tree.

Program 3. Identification of Characteristic Nodes

```
function Analyze($v$)
1. $status(v) = inProcess$
2. if $status(p[v]) = inProgress$ then compute $\mu$ value in cycle $p[v] \rightsquigarrow v$
3. else if $status(p[v]) \neq processed$ then Analyze($p[v]$)
4. $status(v) = processed$
main
1. for $v = 2$ to $n - 1$ do
2.     if $status(v) \neq processed$ then Analyze($v$)
```

The computation of the reverse function is detailed in Program 2. Line 1 can be implemented by means of a recursive function that follows the outgoing edge of each node until it identifies a cycle, then an auxiliary function is used to compute the min/max value in that cycle (see Program 3). This requires $O(n)$ time. An integer sorting algorithm can be used to sort the f_i values in increasing or decreasing order and cycles edges can be inverted as described in the analysis of the encoding algorithm. Line 7 (if required) identifies the only node v in the cycle of f_i such that $p[v] = f_i$, each such node becomes new value for f_i: this require $O(n)$ time. Thus, the overall decoding procedure requires linear time.

5 Parallel Implementation

In this section we present a parallel version of the encoding and decoding algorithms proposed in Sec.4. Our algorithms are described for the theoretical PRAM model and costs are expressed as the number of processors multiplied by the maximum time required by a single processor.

We choose the PRAM classical model because we do not need to address any specific hardware. In the last decade, PRAM model has been deemed useless by many researchers because it is too abstract compared with actual parallel architectures. As noted in the introduction, this trend is changing. At SPAA'07, Vishkin and Wen reported about the recent advancements achieved at the University of Maryland within the project PRAM-On-Chip [23]. The XMT (eXplicit Multi-Threading) general-purpose computer architecture is a promising parallel algorithmic architecture to implement PRAM algorithms. They also developed a single-instruction multiple-data (SIMD) multi-thread extension of C language with the intent of providing an easy programing tool to implement PRAM algorithms. It has primitives like: Prefix Sum, Join, Fetch and Increment, etc. Thus we think that PRAM is robust, reasonable, and well studied theoretical framework for describing high level parallel algorithms.

In the following we will consider PRAM with Exclusive Write (EW) and either Exclusive Read (ER) or Concurrent Read (CR). Due to the lack of space, we don't explicitly recall the well known basic techniques used in this section; we refer the interested reader to [24].

Parallel Encoding. We now describe how to encode Dandelion-like codes on an EREW PRAM; the parallel algorithm is detailed in Program 4.

Program 4. Dandelion-Like Parallel Encoding Algorithm

Parameters: μ, $\upharpoonleft\downharpoonleft$, invertEdges
Input: a tree T represented by its parent vector p
Output: a functional digraph G_p such that $p[0]$ is undefined and $p[1] = 0$

```
1.  Compute $\min(T_v)$ and $level(v)$ for each $v \in T$
2.  Create $P$ corresponding to the path $1 \rightsquigarrow 0$
3.  Compute $\mu_{\upharpoonleft\downharpoonleft}(v)$ for each $v \in P$ excluding 1 and 0
4.  Identify all nodes $f_1, f_2, \ldots, f_k$ such that $\mu_{\upharpoonleft\downharpoonleft}(f_i) = f_i$
5.  $f_0 = 1$; $f_{k+1} = 0$
6.  if $\upharpoonleft\downharpoonleft = \text{down}$ then
7.      for $i = 1$ to $k$ in parallel do $f_i = pred[f_{i+1}]$
8.  for $i = 1$ to $k$ in parallel do $p[f_i] = p[f_{i-1}]$
9.  $p[1] = 0$
10. if invertEdges then
11.     for $v \in P$ in parallel do $p[p[v]] = v$
```

Initially we compute, for each node v, $\min(T_v)$: the minimum value in the subtree rooted at v. If $\min(T_v) = 1$ then v is in the path $1 \rightsquigarrow 0$. This operation requires $O(\log n)$ time with $O(n/\log n)$ processors by using the Rake technique. With the same bounds we can compute the top-down level of each node (Euler Tour technique): exploiting this information we are able to create a vector P containing the sequence of all nodes in the path $1 \rightsquigarrow 0$.

The function $\mu_{\upharpoonleft\downharpoonleft}(v)$ can be computed for all nodes in P (with the same time and processors bounds of the above operations) regarding this path as a tree T_P and computing the max/min value in the subtree of each node. If $\upharpoonleft\downharpoonleft = \text{down}$ then T_P have to be rooted in 0, otherwise it must be rooted in 1. To order the f_i values we can use Prefix-Sum on vector P assigning 1 to nodes v such that $\mu_{\upharpoonleft\downharpoonleft}(v) = v$ and 0 otherwise.

The three cycles of lines 7, 8, and 11 require O(1) with n processors and do not imply concurrent reading or writing. The value $pred[v]$ (the predecessor of v in the path $1 \rightsquigarrow 0$) can be computed in the following way: for each node in $v \in P$ set $pred[p[v]] = v$. Applying Brent's Theorem all these operations can be scheduled on $O(n/\log n)$ processors in $O(\log n)$ time. The overall cost is linear and thus the algorithm is optimal.

Parallel Decoding. The most demanding step in the parallel decoding algorithm is the computation of characteristic nodes. It can be obtained in $O(\log n)$ time with $O(n)$ processors on a CREW PRAM in a Pointer Jumping like fashion: for each node we follow the outgoing edge searching for the max/min value in the ascending path (the parameter μ discriminate whether to search for maximum or minimum values). After each step we set $p[v] = p[p[v]]$, thus obtaining a single Pointer Jump. The procedure is detailed in Program 5: the value $\texttt{asc}(v)$ is the min/max value in the ascending path of v. At each step $\texttt{asc}(v)$ is compared with $\texttt{asc}(p[v])$ and eventually updated. Notice that we explicitly flag whether the value $\texttt{asc}(v)$ comes from v itself or has been encountered in the proper ascending path. At the end of the $\log n$ iterations if $\text{asc}(v) = v$ and $\text{self}(v)$ is false, we can state that v is the min/max

Program 5. Parallel Identification of Characteristic Nodes

```
1. $p[0] = 0$
2. for each node $v \in T$ in parallel do asc($v$) = $v$; self($v$) = true
3. for $j = 1$ to $\lceil \log n \rceil$ do
4.    for each node $v \in T$ in parallel do
5.       if asc($p[v]$) = $\mu${asc($p[v]$), asc($v$)} then
6.          asc($v$) = asc($p[v]$); self($v$) = false
7.       $p[v] = p[p[v]]$
```

Program 6. Dandelion-Like Parallel Decoding Algorithm

Parameters: μ, $\upharpoonleft\downharpoonright$, INVERTEDGES
Input: a functional digraph G_p such that $p[0]$ is undefined and $p[1] = 0$
Output: a tree T represented by its parent vector p

```
1. Identify all characteristic nodes $f_1, f_2, \ldots, f_k$ according with $\mu$ and $\upharpoonleft\downharpoonright$
2. if INVERTEDGES then
3.    for each $v \in$ cycles in parallel do $p[p[v]] = v$
4. $f_0 = 1$; $f_{k+1} = 0$
5. if $\upharpoonleft\downharpoonright$ = down then
6.    for $i = 1$ to $k$ in parallel do $f_i = pred(f_i)$
7. for $i = 0$ to $k - 1$ in parallel do
8. $p[f_i] = p[f_{i+1}]$
9. $p[f_k] = 0$
```

node in its cycle. Indeed, a value equal to v has been found in the proper ascending path of v, this means that there exists a path $v \rightsquigarrow v$, i.e., a cycle. Moreover, no node smaller/greater that v has been encountered in this cycle. All these nodes can be enumerated with Prefix-Sum to generate the (increasing or decreasing) ordered sequence of the characteristic nodes $f_1, f_2, \ldots, f_k$. We assume that a copy of p is used in Program 5, since the original vector p will be required latter in the decoding.

Once characteristic nodes have been identified, a further Pointer Jumping can be used to broadcast a flag in their ascending paths, thus identifying all nodes belonging to some cycle. The rest of the Decoding Algorithm proceeds as detailed in Program 6. Namely, all the three cycles of lines 3, 6, and 8 require O(1) time with n processors, provided that the *pred* values are computed for all nodes belonging to any cycle (as described in Parallel Encoding for nodes in P).

6 Conclusions and Open Problems

Concluding, we want to remark that also for the Blob code [11,14] it is possible to obtain parallel algorithms with the very same costs of those presented in this paper. This result can be obtained combining ideas presented in this paper with the redefinition of the Blob code in terms of transformation of trees in to functional digraphs given in [7]. With respect to Table 1, it remains an open problem to study parallel algorithms for the Chen code.

References

1. Boppana, V., Hartanto, I., Fuchs, W.: Full Fault Dictionary Storage Based on Labeled Tree Encoding. In: IEEE VTS 1996, pp. 174–179 (1996)
2. Garg, V., Agarwal, A.: Distributed Maintenance of a Spanning Tree Using Labeled Tree Encoding. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 606–616. Springer, Heidelberg (2005)
3. Deo, N., Kumar, N., Kumar, V.: Parallel Generation of Random Trees and Connected Graphs. Congr. Num. 130, 7–18 (1998)
4. Reeves, C., Rowe, J.: Genetic Algorithms: A Guide to GA Theory. Springer, Heidelberg (2003)
5. Cayley, A.: A Theorem on Trees. Quart. J. Math. 23, 376–378 (1889)
6. Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. Archiv der Mathematik und Physik 27, 142–144 (1918)
7. Caminiti, S., Petreschi, R.: String Coding of Trees with Locality and Heritability. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 251–262. Springer, Heidelberg (2005)
8. Chen, W.: A General Bijective Algorithm for Increasing Trees. Syst. Sci. Math. Sci. 12, 194–203 (1999)
9. Deo, N., Micikevičius, P.: A New Encoding for Labeled Trees Employing a Stack and a Queue. Bulletin of ICA 34, 77–85 (2002)
10. Eğecioğlu, Ö., Remmel, J.: Bijections for Cayley Trees, Spanning Trees, and Their q-Analogues. J. Comb. Th. 42A, 15–30 (1986)
11. Kreweras, G., Moszkowski, P.: Tree codes that preserve increases and degree sequences. J. Disc. Math. 87, 291–296 (1991)
12. Moon, J.: Counting Labeled Trees. William Clowes and Sons, London (1970)
13. Neville, E.: The Codifying of Tree-Structure. In: Proc. of Cambridge Philosophical Soc., vol. 49, pp. 381–385 (1953)
14. Picciotto, S.: How to Encode a Tree. PhD thesis, U. California, San Diego (1999)
15. Caminiti, S., Finocchi, I., Petreschi, R.: On Coding Labeled Trees. TCS 382, 97–108 (2007)
16. Caminiti, S., Deo, N., Micikevičius, P.: Linear-time Algorithms for Encoding Trees as Sequences of Node Labels. Congr. Num. 183, 65–75 (2006)
17. Chen, H., Wang, Y.: An Efficient Algorithm for Generating Prüfer Codes from Labelled Trees. TCS 33, 97–105 (2000)
18. Deo, N., Micikevičius, P.: Parallel Algorithms for Computing Prüfer-Like Codes of Labeled Trees. Technical report, CS-TR-01-06, Department of Computer Science, University of Central Florida, Orlando (2001)
19. Greenlaw, R., Halldórsson, M., Petreschi, R.: On Computing Prüfer Codes and Their Corresponding Trees Optimally in Parallel. In: JIM 2000 (2000)
20. Caminiti, S.: On Coding Labeled Trees. PhD thesis, Sapienza U. of Rome (2007)
21. Paulden, T., Smith, D.: Recent advances in the study of the dandelion code, happy code, and blob code spanning tree representations. In: IEEE CEC 2006, pp. 2111–2118 (2006)
22. Gottlieb, J., Julstrom, B., Raidl, G., Rothlauf, F.: Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search. In: GECCO 2001, pp. 343–350 (2001)
23. Wen, X., Vishkin, U.: PRAM-on-Chip: First Commitment to Silicon. In: ACM SPAA 2007, pp. 301–302 (2007)
24. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Reading (1992)

Deterministic Computation of Pseudorandomness in Sequences of Cryptographic Application*

A. Fúster-Sabater[1], P. Caballero-Gil[2], and O. Delgado-Mohatar[1]

[1] Institute of Applied Physics, C.S.I.C., Serrano 144, 28006 Madrid, Spain
amparo@iec.csic.es, oscar.delgado@iec.csic.es
[2] Fac. of Maths, D.E.I.O.C., University of La Laguna, 38271 Tenerife, Spain
pcaballe@ull.es

Abstract. An easy method of checking balancedness degree as well as run quantification in sequences obtained from LFSR-based keystream generators has been developed. The procedure is a deterministic alternative to the traditional application of statistical tests. The computation method allows one to check deviation of balancedness and run distribution goodness from standard values. The method here developed can be considered as a first selective criterium for acceptance/rejection of this type of generators of cryptographic application.

Keywords: Pseudorandomness, bit-string, balancedness, run dirtribution, cryptography.

1 Introduction

Transmission of sensitive information between two interested parties needs several security requirements (confidentiality, integrity, non repudiation, authentication ...) that can be satisfied by means of design, assessment and implementation of cryptographic algorithms and security protocols.

Confidentiality makes use of an encryption function currently called *cipher* that converts the *plaintext* into the *ciphertext*. Ciphers are usually divided into two large classes: stream ciphers and block-ciphers. Stream ciphers are very fast (in fact, the fastest among the encryption procedures) so they are implemented in many technological applications e.g. algorithms A5 in GSM communications [7] or the encryption system E0 used in the Bluetooth specifications [1]. Stream ciphers try to imitate the ultimate one-time pad cipher and are supposed to be good pseudorandom generators capable of stretching a short secret seed (the secret key) into a long sequence of seemingly random bits (the keystream sequence). This sequence is then XORed with the plaintext in order to obtain the

* Work supported by CDTI (Spain) and the companies INDRA, Unión Fenosa, Tecnobit, Visual-Tools, BrainStorm, SAC and TechnoSafe under programme Cenit-HESPERIA as well as supported by Ministry of Science and Innovation and European FEDER Fund under Project TIN2008-02236/TSI.

ciphertext. Most generators producing keystream sequence are based on Linear Feedback Shift Registers (LFSRs) e.g. see [6] and [8]. The pseudorandom output sequence is a periodic sequence generated as the image of a nonlinear Boolean function in the LFSR stages.

Desirable properties for such sequences can be enumerated as follows: 1) large period, 2) large linear complexity, 3) good statistical properties. There are well-known proposals [2], [4], [9], [11], [12] for which conditions 1) and 2) above are perfectly satisfied. Nevertheless, how to generate sequences with good statistics is a feature that even now remains quite diffuse, see [3].

Balancedness and adequate distribution of $1's$ and $0's$ in the output sequence are necessary (although not sufficient) conditions that every keystream generator must satisfy. Roughly speaking, a binary sequence is balanced if it has approximately the same number of $1's$ as $0's$. On the other hand, a run of $1's$ ($0's$) of length k is defined as a succession of k consecutive $1's$ ($0's$) between two $0's$ ($1's$). The runs of $1's$ are called *blocks* while the runs of $0's$ are called *gaps*. It is a well known fact [6] that in a pseudorandom binary sequence of period T there are $T/2$ runs distributed as follows: half the runs have length 1, one quarter of the runs length 2, one eighth of the runs length 3, and so forth. Moreover, half the runs of any length are gaps, the other half are blocks. That is to say, in a pseudorandom binary sequence the number and distribution of digits is perfectly quantified.

Due to the long period of the output sequence ($\simeq 10^{77}$) in cryptographic applications, it is unfeasible to produce an entire cycle of such a sequence and then analyze the number and distribution of $1's$ and $0's$. Therefore, in practice, portions of the output sequence are chosen randomly and different statistical tests [11](monobit test, run test, poker test, serial test ...) are applied to all these subsequences. Nevertheless, passing the previous tests merely provides *probabilistic* evidence that the LFSR-based keystream generator produces a sequence with certain characteristics of pseudorandomness.

In this work, *deterministic* expressions to compute the degree of balancedness and number of runs in one period of the output sequence are proposed. If the computed values are not in the expected range, then the generator must be rejected.

2 Notation and Basic Concepts

Any L-variable Boolean function can be expressed canonically in terms of its *minterms* [5], that is the logic product of the L variables (a_1, a_2, ..., a_L) where each variable can be in its true or complementary form. Examples of minterms of L variables are:

$$a_1 a_2 \ldots a_L, \qquad \overline{a}_1 a_2 \ldots \overline{a}_L, \qquad \overline{a}_1 \overline{a}_2 \ldots \overline{a}_L$$

where the superposition of variables represents the logic product. In addition, any L-variable Boolean function can be uniquely expressed in *Algebraic Normal Form* (A.N.F.) or *Muller expansion* [10] by means of the sum exclusive-OR of logic products of different orders in the L variables. A simple example of Boolean function in A.N.F. is:

$$f(a_1,\, a_2,\, ...,\, a_L) = a_1 a_2 \oplus a_2 a_{L-1} \oplus a_L$$

where $\oplus$ represents the exclusive-OR logic operation.

In mathematical terms, a LFSR-based generator is a L-variable nonlinear Boolean function, $F : GF(2)^L - \{0\} \to GF(2)$, whose L input-variables are the stages of the LFSRs. At each clock pulse the LFSRs generate new stage contents that will be the new input-variables of F. In this way, the generator produces the successive bits of the *output sequence* or *generated sequence*. A LFSR-based keystream generator is nothing but a nonlinear Boolean function F given in its A.N.F. Moreover, the LFSRs involved in this kind of generator are *maximal length*-LFSRs [6], that is LFSRs whose characteristic polynomials are primitive. In this case, their output sequences are called *PN-sequences*. Balancedness and run distribution of PN-sequences have been extensively studied in the literature. See for example [6] and [13].

Let A be an arbitrary *maximal length*-LFSR of length L_A and a_i $(i = 1, ..., L_A)$ the binary content of the *i-th* LFSR stage. A minterm of L_A variables is denoted by $A_{i...j}$ whether such a minterm includes the variables a_i ... a_j in their true form while the other variables are in complementary form.

Let Λ_L denote the set of L-variable Boolean functions in A.N.F. and Φ_F the *minterm function* of F. In fact, $\Phi_F : \Lambda_L \to \Lambda_L$, such that, given F, Φ_F substitutes every term of F by its corresponding minterm. For a nonlinear function in A.N.F., e.g. $F(a_1,\, a_2,\, a_3) = a_1\, a_2\, a_3 \oplus a_1\, a_3 \oplus a_2 \oplus a_3$, we have:

$$\Phi_F = A_{123} \oplus A_{13} \oplus A_2 \oplus A_3 .$$

On the other hand, every minterm considered as a generator applied to the L_A stages of A generates a canonical sequence with an unique 1 and period $T = 2^{L_A} - 1$, see [11].

Once the nonlinear function F given in its A.N.F. has been converted into its minterm expansion, the basic ideas of this work can be summarized as follows:

1. The number of minterms in the representation of F equals the number of $1's$ in the output sequence as every minterm provides the generated sequence with an unique 1.
2. The contiguity of such minterms in the ordered minterm succession determines the run distribution in the output sequence.

Let us now generalize the previous statements to more than one LFSR. Let A, $B, ...,$ Z be N LFSRs whose lengths are respectively L_A, L_B, ... , L_Z (supposed $GCD(L_i, L_j) = 1, \quad i \neq j$). We denote by a_i $(i = 1, ..., L_A)$, b_j $(j = 1, ..., L_B)$, ... , z_k $(k = 1, ..., L_Z)$ their corresponding stages. The *global minterms* associated with the generator have now $L_A + L_B + ... + L_Z$ variables and are of the form, e.g. $A_{ij}\, B_{pqr} ... Z_s$, that is to say the ordered product of the individual minterms of each LFSR. As before every global minterm considered as a generator applied to the stages of the LFSRs generates a sequence with an unique 1 and period $T = (2^{L_A} - 1)(2^{L_B} - 1) \ldots (2^{L_Z} - 1)$ [11]. In brief, every LFSR-based generator can be expressed in terms of its global minterms as well as global minterms determine balancedness and run distribution in the output sequence.

3 From A.N.F. to Global Minterm Expansion

Previously to the conversion algorithm, the following facts are introduced:

Fact 1: For every Boolean function F the following equality holds [10]

$$F = \Phi_F \circ \Phi_F \tag{1}$$

where the symbol $\circ$ denotes the composition of functions.

Fact 2: For every LFSR A, the exclusive-OR of all the minterms [11] is:

$$A_{12 \ldots L_A} \oplus A_{12 \ldots L_A - 1} \oplus \ldots \oplus A_{2 \ldots L_A} \oplus \ldots \oplus A_{L_A} \oplus \ldots \oplus A_2 \oplus A_1 = 1. \tag{2}$$

The previous equation can be rewritten as:

$$A_1^{'} \oplus A_1 = 1. \tag{3}$$

On the other hand, the total number of terms in (2) is:

$$\sum_{i=1}^{L_A} \binom{L_A}{i} = 2^{L_A} - 1 \tag{4}$$

while the number of terms in $A_1 = a_1 \overline{a}_2 \ldots \overline{a}_{L_A}$ is [10]:

$$N_t\ (A_1) = 2^{L_A - 1}. \tag{5}$$

Thus, the number of terms in $A_1^{'}$ will be:

$$N_t\ (A_1^{'}) = 2^{L_A - 1} - 1. \tag{6}$$

Appropriate notation will be used for the rest of LFSRs.

3.1 Conversion Algorithm

Input: N (number of LFSRs), L_A, L_B, ..., L_Z (lengths of the LFSRs) and a nonlinear function F in A.N.F.

For instance, $N_Z = 2$, $L_A = 2$, $L_B = 3$ and $F(a_1, a_2, b_1, b_2, b_3) = a_1\, b_1$.

– *Step 1:* Compute Φ_F

$$\Phi_F = A_1\, B_1.$$

– *Step 2:* Substitute every minterm by its corresponding function in A.N.F. and cancel common terms (if there exist)

$$\Phi_F = (a_1\, a_2 \oplus a_1)(b_1\, b_2\, b_3 \oplus b_1\, b_2 \oplus b_1\, b_3 \oplus b_1) =$$
$$a_1\, a_2\, b_1\, b_2\, b_3 \oplus a_1\, a_2\, b_1\, b_2 \oplus a_1\, a_2\, b_1\, b_3 \oplus a_1\, a_2\, b_1 \oplus$$
$$a_1\, b_1\, b_2\, b_3 \oplus a_1\, b_1\, b_2 \oplus a_1\, b_1\, b_3 \oplus a_1\, b_1 .$$

– *Step 3:* Compute $F(a_i,\, b_j) = \Phi_F \circ \Phi_F$

$$F(a_i,\, b_j) = \Phi_F \circ \Phi_F = A_{12}\, B_{123} \oplus A_{12}\, B_{12} \oplus A_{12}\, B_{13} \oplus A_{12}\, B_1 \oplus$$
$$A_1\, B_{123} \oplus A_1\, B_{12} \oplus A_1\, B_{13} \oplus A_1\, B_1 .$$

Output: F expressed in terms of its global minterms.

Once the function F has been expressed in terms of its minterms, balancedness and run distribution in the output sequence can be analyzed.

4 Balancedness in the Keystream Sequence

The number of 1's in the generated sequence coincides with the number of global minterms in the expression of F or, equivalently, the number of terms in Φ_F (*Step 2*). An illustrative example of application is presented.

4.1 A Numerical Example

Example 1: Let A, B, C be three LFSRs of lengths L_A, L_B, L_C respectively. The *generating function* is chosen:

$$F = a_1 b_1 \oplus b_1 c_1 \oplus c_1 \tag{7}$$

(Geffe's generator [13]) and the *minterm function* Φ_F is computed:

$$\begin{aligned}\Phi_F &= A_1 B_1 \oplus B_1 C_1 \oplus C_1 \\ &= A_1 B_1 (C_1' \oplus C_1) \oplus (A_1' \oplus A_1) B_1 C_1 \oplus \\ &\quad \oplus (A_1' \oplus A_1)(B_1' \oplus B_1) C_1 \\ &= A_1 B_1 (C_1' \oplus C_1) \oplus (A_1' \oplus A_1) B_1' C_1 \,.\end{aligned}$$

The number of $1's$ in the output sequence can be directly obtained by counting the number of terms in Φ_F via the equations (5) and (6). According to this simple rule, the number of $1's$ in the output sequence obtained from a Geffe's generator is given by:

$$No.\ 1's = 2^{L_A - 1} 2^{L_B - 1} (2^{L_C} - 1) + (2^{L_A} - 1)(2^{L_B - 1} - 1) 2^{L_C - 1}. \tag{8}$$

Remark that the previous expression is function exclusively of the lengths of the LFSRs. In a practical range, we say $L_i \simeq 120$, and keeping in mind that the sequence period is $T = (2^{L_A} - 1)(2^{L_B} - 1)(2^{L_C} - 1)$, the number of $1's$ in the output sequence is:

$$No.\ 1's \simeq T/2.$$

Consequently, the generated sequence is balanced.

5 Run Distribution in the Keystream Sequence

The computation of runs in the output sequence is based on the following result.

Proposition 1. *Let us consider the ordered minterm succession of a maximal length-LFSR of length L. If the minterms including an arbitrary index i are replaced by* 1 *and the minterms not including the index i are replaced by* 0*, then the resulting binary sequence is the reverse version of the PN-sequence generated by the LFSR.*

The previous result is a straight application of the LFSR linear recurrence relationship given by its characteristic polynomial. Thus, a minterm succession can be treated as a PN-sequence. Now, keep in mind the following remarks:

1. The number of runs of any length of a PN-sequence is perfectly quantified. Indeed, each m-gram (every one of the 2^m possible configurations of m bits $(m = 1, ..., L)$) will appear exactly 2^{L-m} times throughout the PN-sequence except for the L-gram $00\ldots0$ that will not appear any time.
2. In the global minterm succession each m-gram of any LFSR will coincide once with each one of the m-grams of the other LFSRs.

Based on these considerations, the computation of runs in the output sequence can be carried out as it is shown in the following example.

Example 2: For two LFSRs, A and B, of lengths L_A and L_B respectively (L_A $< L_B$) and generating function $F = a_1b_1$, we proceed:

$$\Phi_F = A_1B_1 = (a_1 \oplus a_1a_2 \oplus \ldots \oplus a_1a_2\ldots a_{L_A})(b_1 \oplus b_1b_2 \oplus \ldots \oplus b_1b_2\ldots b_{L_B})$$

$$F = \Phi_F \circ \Phi_F = (A_1 \oplus A_{12} \oplus \ldots \oplus A_{12\ldots L_A})(B_1 \oplus B_{12} \oplus \ldots \oplus B_{12\ldots L_B})$$

Notice that the minterm expansion of F will only include products of individual minterms with the index 1. Let us now introduce the following notation:

Y denotes an arbitrary minterm of A or B including the index 1.

N denotes an arbitrary minterm of A or B not including the index 1.

$SecA$ denotes the ordered succession of minterms of A in format Y/N.

$SecB$ denotes the ordered succession of minterms of B in format Y/N.

It is clear that an 1 in the output sequence corresponds to a minterm product YY (for example, A_1B_{12}) while a 0 in the output sequence corresponds to the minterm products YN, NY or NN (for example, $A_1B_2, A_{23}B_{13}$ or A_3B_2). See the formation rule in Table 1.

Now we can compute the number of runs of different lengths.

Table 1. Global minterm formation rule for F in Example 1

SecB	SecA	Output bit
Y	Y	1
N	Y	0
Y	N	0
N	N	0

5.1 Runs of Length 1

Blocks: They are runs of the form "0 1 0" that come from minterm structures

$$SecB : * \, Y \, * \\ SecA : * \, Y \, *$$

The symbol $*$ denotes Y or N. The 3-gram NYN will appear 2^{L_A-3} times in $SecA$ and 2^{L_B-3} times in $SecB$, the 2-gram $NY*$ will appear 2^{L_A-2} times in $SecA$ and 2^{L_B-2} times in $SecB$, and so forth.

The different configurations of minterms able to generate a block of length 1 are depicted in Table 2 at columns with heading "Configurations". The columns with heading "No. of config." show the number of times that such configurations appear on their corresponding minterm sequences.

Thus, the number of blocks of length 1 will be the sum of all suitable configurations multiplied by the number of times that such configurations appear

$$N_B(1) = (2^{L_B-1} + 2 \cdot 2^{L_B-2} + 2^{L_B-3})\, 2^{L_A-3} \; . \tag{9}$$

Gaps: They are runs of the form "1 0 1" that come from minterm structures

$$SecB : Y \, * \, Y \\ SecA : Y \, * \, Y$$

The different configurations of minterms able to generate a gap of length 1 are depicted in Table 3.

Thus, the number of gaps of length 1 will be the sum of all suitable configurations multiplied by the number of times that such configurations appear

$$N_G(1) = (2^{L_B-3} + 2^{L_B-2})\, 2^{L_A-3} \; . \tag{10}$$

Table 2. Configurations of minterms producing blocks of length 1

	Configuration	No. of config.	Configuration	No. of config.
SecB	$* \, Y \, *$	2^{L_B-1}	$* \, Y \, N$	2^{L_B-2}
SecA	$N \, Y \, N$	2^{L_A-3}	$N \, Y \, Y$	2^{L_A-3}
SecB	$N \, Y \, *$	2^{L_B-2}	$N \, Y \, N$	2^{L_B-3}
SecA	$Y \, Y \, N$	2^{L_A-3}	$Y \, Y \, Y$	2^{L_A-3}

Table 3. Configurations of minterms producing gaps of length 1

	Configuration	No. of config.	Configuration	No. of config.
SecB	$Y \, N \, Y$	2^{L_B-3}	$Y \, * \, Y$	2^{L_B-2}
SecA	$Y \, Y \, Y$	2^{L_A-3}	$Y \, N \, Y$	2^{L_A-3}

5.2 Runs of Length n

The procedure can be generalized in order to compute the number of runs of length n ($n = 1, ..., L_A - 2$).

Blocks: They are runs of the form "0 1 . . . 1 0" (with n consecutive $1's$) coming out from minterm structures

$$SecB : * \, Y \dots \, Y \; * \\ SecA : * \, Y \dots \, Y \, *$$

with n minterms Y in both sequences. The different configurations able to generate a block of length n and their number are depicted in Table 4.

Thus, the number of blocks of length n will be:

$$N_B(n) = (2^{L_B - (n+2)} + 2 \cdot 2^{L_B - (n+1)} + 2^{L_B - n})\, 2^{L_A - (n+2)} \quad . \tag{11}$$

Gaps: They are runs of the form "1 0 . . . 0 1" (with n consecutive $0's$) coming out from minterm structures

$$SecB : Y \; * \dots * \; Y \\ SecA : Y \; * \dots * \; Y$$

with n symbols $*$ in both sequences. Notice that in $SecA$ there will be 2^n different configurations able to generate a gap of length n ranging from $Y \, N \; \dots \; N \, Y$ up to $Y \, Y \; \dots \; Y \, Y$. Some of such configurations and their number are depicted in Table 5.

Thus, the number of gaps of length n will be:

$$N_G(n) = (\sum_{i=0}^{n} \binom{n}{i} \, 2^{L_B - (n+2-i)} \;) \cdot 2^{L_A - (n+2)}. \tag{12}$$

Therefore, the number of runs of any length up to $L_A - 2$ can be easily computed in the proposed example. Equations (11) and (12) give us the exact

Table 4. Configurations of minterms producing blocks of length n

	Configuration	No. of config.	Configuration	No. of config.
SecB	$N\,Y \dots Y\,N$	$2^{L_B - (n+2)}$	$*\;Y \dots Y\,N$	$2^{L_B - (n+1)}$
SecA	$Y\,Y \dots Y\,Y$	$2^{L_A - (n+2)}$	$N\,Y \dots Y\,Y$	$2^{L_A - (n+2)}$
SecB	$N\,Y \dots Y\,*$	$2^{L_B - (n+1)}$	$*\,Y \dots Y\;*$	$2^{L_B - n}$
SecA	$Y\,Y \dots Y\,N$	$2^{L_A - (n+2)}$	$N\,Y \dots Y\,N$	$2^{L_A - (n+2)}$

Table 5. Configurations of minterms producing gaps of length n

	Configuration	No. of config.	Configuration	No. of config.
SecB	$Y\; * \dots * \;Y$	$2^{L_B - 2}$	$Y\,N \dots N\,Y$	$2^{L_B - (n+2)}$
SecA	$Y\,N \dots N\,Y$	$2^{L_A - (n+2)}$	$Y\,Y\; \dots Y\,Y$	$2^{L_A - (n+2)}$

Table 6. Numerical example

n	No. of blocks	%Deviation(blocks)	No. of gaps	%Deviation(gaps)
1	4608	13,8 %	1536	62,0 %
2	1152	43,1 %	1152	43,1 %
3	288	71,5 %	864	14,6 %
4	72	85,7 %	648	28,1 %
5	18	92,8 %	486	92,1 %

number of blocks and gaps that can be found in the output sequence. Remark that N_B and N_G depend exclusively on the LFSR's lengths L_A and L_B and on the run length (n). There is no dependency on the primitive characteristic polynomials. Consequently, different LFSRs of the same length primitive characteristic polynomials will produce output sequences with the same number of blocks and gaps.

According to these expressions, it can be seen that the analyzed function F does not match the expected values. For a numerical example $L_A = 7$ and $L_B = 8$, see results in Table 6. For $n = 1$, $N_B > N_G$. For $n = 2$, equations (11) and (12) coincide. For $n \geq 3$, $N_B < N_G$. As expected, there are more gaps than blocks because the formation rule in Table 1 is not balanced.

The upper limit $L_A - 2$, L_A being the length of the shortest LFSR, follows from the fact that blocks and gaps of length n include $n + 2$ bits but we can only guarantee the presence of at most L_A-grams. At any rate, the designer of keystream generators is basically interested in the runs of low length (e.g. up to length 15) while in, for instance, cryptographic applications every LFSR length takes values in the range $L_i \simeq 120$.

6 Conclusions

An easy and efficient method of checking the degree of balancedness and number of runs in the output sequence of keystream generators has been presented. From the concept of global minterm, it is possible to derive general expressions for the number of $1's$ $(0's)$ as well as for the number of runs in the output sequence of any LFSR-based generator. Then, the obtained values are compared with the expected values for a sequence to be pseudorandom. The result of this comparison implies the assessment of such a sequence generator.

In this work, such a method has been applied exclusively to nonlinear generating functions. Nevertheless, these ideas concerning the analysis of the global minterms seem to be suitable for more general pattern generators. Consider, for instance, the *multiple-speed generators* [11] that can be expressed in terms of a more complex generating function or the *shrinking generator* [11] whose global minterms can be obtained by removing certain individual minterms from one of the LFSRs. In both cases, the developed method can be adapted and applied to these schemes in order to evaluate certain aspects of pseudorandomness in the generated sequences.

References

1. Bluetooth, Specifications of the Bluetooth system, Version 1.1 (February 2001), http://www.bluetooth.com/
2. Caballero-Gil, P., Fúster-Sabater, A.: A Wide Family of Nonlinear Filter Functions with a Large Linear Span. Inf. Sci. 164, 197–207 (2004)
3. Fúster-Sabater, A., García-Mochales, P.: On the Balancedness of nonlinear generators of binary sequences. Inf. Process. Lett. 85, 111–116 (2003)
4. Fúster-Sabater, A., Caballero-Gil, P.: Linear solutions for cryptographic nonlinear sequence generators. Phys. Lett. A. 369, 432–437 (2007)
5. Grassman, W.K., Trembley, J.P.: Logic and Discrete Mathematics. Prentice-Hall, New York (1996)
6. Golomb, S.W.: Shift Register-Sequences. Aegean Park Press, Laguna Hills (1982)
7. GSM, Global Systems for Mobile Communications, http://cryptome.org/gsm-a512.htm
8. Kanso, A.: Clock-Controlled Shrinking Generator of Feedback Shift Registers. In: Safavi-Naini, R., Seberry, J. (eds.) ACISP 2003. LNCS, vol. 2727, pp. 443–451. Springer, Heidelberg (2003)
9. Klapper, A.: Large Families of Sequences with Near Optimal Correlations and Large Linear Span. IEEE Trans. Inf. Theory. 42, 1241–1248 (1996)
10. Mange, D.: Analysis and Synthesis of Logic Systems. Artech House, Inc., Norwood (1986)
11. Menezes, A.J., et al.: Handbook of Applied Cryptography. CRC Press, New York (1997)
12. Shparlinski, I.E.: On the Linear Complexity of the Power Generator. Design, Codes and Cryptography 23, 5–10 (2001)
13. Simmons, G.J. (ed.): Contemporary Cryptology: The Science of Information Integrity. IEEE Press, New York (1991)

Parallel Simulated Annealing for the Job Shop Scheduling Problem

Wojciech Bożejko[⋆], Jarosław Pempera, and Czesław Smutnicki

Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl, jaroslaw.pempera@pwr.wroc.pl,
czeslaw.smutnicki@pwr.wroc.pl

Abstract. This paper describes two parallel simulated annealing algorithms for the job shop scheduling problem with the sum of job completion times criterion. Some properties of the problem associated with the *block theory* have been presented and discussed. These properties allow us to introduce the effective neighborhood based on the adjacent swap type moves. In this paper, an original method for parallel calculation of optimization criterion value for set of solutions, recommended for the use in metaheuristics with single- and multiple- search trajectories is proposed. Additionally, the vector calculation method, that uses multiple mathematical instructions MMX supported by suitable data organization, is presented. Properties of parallel calculations are empirically verified on the PC with Intel Core 2 Duo processor on Taillard's benchmarks.

Keywords: parallel metaheuristics, scheduling, optimization, job shop, simulated annealing.

1 Introduction

Job shop scheduling problems follow from many real cases, which means that they own good practical applications as well as the industrial significance. Because of NP-hardness of the problem, despite the criteria value form, heuristics and metaheuristics are recommended as "the most reasonable" solution methods. The majority of these methods refers to the makespan minimization. We mention here, as an example, a few recent studies: Jain, Rangaswamy, and Meeran [7]; Pezella and Merelli [11]; Grabowski and Wodecki [4]; Nowicki and Smutnicki [10].

The job shop problem with general regular criteria is commonly regarded as *harder* than the job shop problem with the makespan criterion, mainly because of the lack of special properties that would reinforce the solution algorithm. Moreover, till now, there have not been discovered for the problem any *sequential accelerators* (properties that can speed up computations throughout skillful aggregation and decomposition of calculations). In this context, parallelization

[⋆] Corresponding author.

techniques are the only methods allowing ones, practical instances in reasonable time to be solved. Therefore they are especially desirable.

Some heuristics algorithms based on dispatching rules for the considered problem are presented in papers of Holthaus and Rajendran [6], Bushee and Svestka [3]. For the other regular criteria such as the total tardiness there are proposed metaheuristics based on various local search techniques: simulated annealing [5], [14], tabu search [2] and genetic search [8].

In this paper we propose a genuine method of cost function computing in parallel by using multi-processor system as well as a single-processor system with multiple mathematical instructions MMX. The obtained results can be applied directly to modern PCs equipped with a few processors, multi-core processors or processors with multiple mathematical instructions.

2 The Problem

We consider a manufacturing system with any structure consisting of m machines of a unit capacity given by the set $M = \{1, \ldots, m\}$. In the system, there are processed n jobs given by set $J = \{1, 2, ..., n\}$. The job j-th requires the sequence of n_j operations indexed consecutively $(l_{j-1} + 1, ..., l_{j-1} + n_j)$, where $l_j = \sum_{i=1}^{j} n_i$, is the total number of operations of the first j jobs, $j = 1, 2, ..., n$, $(l_0 = 0)$, and $o = \sum_{i=1}^{n} o_i$. The operation x is to be processed on the machine $\mu_x \in M$ during an uninterrupted processing time $p_x > 0$, $x \in O = \{1, 2, \ldots, o\}$. Our aim is to find the schedule under the following constraints: (1) each machine can process at most one product at a time, (2) each product can be processed by at most one machine at a time, (3) operations cannot be preempted.

The set of operations O can be decomposed into subsets $O_k = \{x \in O : \mu_x = k\}$, each of them contains operations to be processed on the machine k, $k \in M$. Let the permutation π_k defines the processing order of operations from the set O_k on machine k, and let Π_k be the set of all permutations on O_k. The processing order of all operations on machines is determined by the m-tuple $\pi = (\pi_1, \pi_2, ..., \pi_m)$, where $\pi \in \Pi_1 \times \Pi_2 \times ... \times \Pi_m$.

For any operation $j \in O$ and processing order given by π we define the machine predecessor/successor $\underline{s}_j$, $\overline{s}_j$ as well as the technological predecessor/successor $\underline{t}_j$, $\overline{t}_j$, according to the expressions below

$$\underline{s}_{\pi_i(j)} = \begin{cases} 0 & j = 1 \\ \pi_i(j-1) & j = 2, \ldots, n_j, \end{cases} \qquad \underline{s}_{\pi_i(j)} = \begin{cases} \pi_i(j+1) & j = 1, \ldots, n_j - 1, \\ 0 & j = n_j, \end{cases}$$

$$\overline{t}_{\pi_i(j)} = \begin{cases} 0 & j = l_{i-1} + 1, i \in J \\ j - 1 & otherwise, \end{cases} \qquad \overline{t}_{\pi_i(j)} = \begin{cases} 0 & j = l_i, i \in J \\ j + 1 & otherwise, \end{cases}$$

Note that if the index of predecessor of the operation j is 0, then j is the first operation of proper job or/and the first operation processed on the proper machine. Similarly, if the index of successor of the operation j is 0 then j is the last operation of proper job or/and the last operation processed on the proper machine. The machine predecessor/successor depends on π, but for simplicity in a notation we will not express it explicitly.

The schedule for the fixed processing order π is described by event vectors $S = (S_1, \ldots, S_o)$ and $C = (C_1, \ldots, C_o)$, where values S_j and C_j denote the starting time of operation j and its completion time. The schedule has to satisfy the following constraints

$$C_{\underline{t}_j} \leq S_j \qquad \underline{t}_j \neq 0, \;\; j \in O, \tag{1}$$

$$C_{\underline{s}_j} \leq S_j \qquad \underline{s}_j \neq 0, \;\; j \in O, \tag{2}$$

$$C_j = S_j + p_j \qquad j \in O. \tag{3}$$

Because of equation (3), the schedule can be represented by a single event vector, and we use C to this aim. The schedule C, for the fixed π, is feasible if it satisfies conditions (1)–(3). The constraint (1) follows from the technological processing order of operations inside job, whereas (2) from the unit capacity of machines. Our aim is to find the feasible processing order $\pi^* \in \Pi$, so that

$$C_{sum}(\pi^*) = \min_{\pi \in \Pi} C_{sum}(\pi), \tag{4}$$

where $C_{sum}(\pi) = \sum_{i \in O^L}^{n} C_i$ is the sum of jobs completion times and $O^L = \{i : i = l_i, \; i \in O\}$ is the set of the last operations of jobs.

It is convenient to represent the processing order π and the schedule C by using the following direct graph $G(\pi) = (O, R \cup E(\pi))$ with a set of nodes O and a set of arcs $R \cup E(\pi)$, where $R = \{(\underline{t}_j, j) : \underline{t}_j \neq 0, j \in O\}$ $E(\pi) = \{(\underline{s}_j, j) : \underline{s}_j \neq 0, j \in O\}$. The node $j \in O$ represents the j-th operation of a certain job and has the weight p_j. Arcs from the set R represent the processing order of operations in jobs and correspond to the constraint (1), whereas arcs from set $E(\pi)$ represent the processing order of operations on machines and correspond to the constraint (2). All arcs from both subsets have the weight zero. Let us start from some well-known facts.

Property 1. *The processing order π is feasible if $G(\pi)$ does not contain a cycle.*

Consider the graph $G(\pi)$ for a feasible π. Denote by U_i the longest path (i.e. sequence of nodes) going to node i and by d_i the length (including p_i) of U_i, $i \in O$.

Property 2. *For each feasible processing order π, there exists a feasible schedule C of the problem, such that $C_i = d_i$ and each C_i is as small as possible, $i \in O$.*

The path U_i can be written as $U_i = (u_i(1), u_i(2), \ldots, u_i(w_i))$, where $u_i(x) \in O$, $1 \leq x \leq w_i$, and w_i is the number of nodes in this path. Clearly, $u_i(w_i) = i$. Each path U_i, $i \in O$ can naturally be decomposed into several specific subpaths, so that each subpath contains nodes linked by the same type of arcs. We define *block* as the maximal subsequence $B_i^* = (u_i(g_i^*), ..., u_i(h_i^*))$ of U_i such that $\mu_{u_i(g_i^*)} = \mu_{u_i(g_i^*+1)} = \ldots = \mu_{u_i(h_i^*)}$ and $(u_i(j), u_i(j+1)) \in E(\pi)$ for all $i = g_i^*, \ldots, h_i^* - 1$,

$g_i^* < h_i^*$. A block corresponds to a sequence of operations (jobs) processed on the same machine without inserted an idle time. In further considerations we will be interested only in *non-empty* blocks, i.e. containing at least two operations.

Based on the properties of the job shop problem with the makespan criterion [4] one can prove the following properties of the problem.

Theorem 1. *Let $B_i^1, B_i^2, \ldots, B_i^{r_i}$ be decomposition of U_i, $i \in O^L$ for the acyclic graph $G(\pi)$. If the acyclic graph $G(\alpha)$ has been obtained from $G(\pi)$ through the modifications of π so that $C_{sum}(\alpha) < C_{sum}(\pi)$, then in $G(\alpha)$ at least one operation $x \in B_i^k$ is executed on earlier position than an original one in permutation π_{μ_x}, for some $k \in \{1, 2, ..., r_i\}$ and some $i \in O^L$.*

Property 3. *Let $B_i^k = (u_i(g_i^k), ..., u_i(h_i^k))$, $k \in \{1, 2, ..., r_i\}$, $i \in O^L$ for the acyclic graph $G(\pi)$. If the graph $G(\alpha)$ has been obtained from $G(\pi)$ through the interchanging two successive operations of B_i^k that $G(\alpha)$ is acyclic.*

Property 4. *All paths U_i, $i \in O$ and their lengths can be found in the time $O(o)$.*

3 Simulated Annealing

Simulated annealing (SA) method applies an analogy to the thermodynamic cooling process to avoid local minima and escape from them. The search trajectory is guided through the set of solution Π in a "statistically suitable" direction. The SA application to our problem is described briefly as follows. In each iteration the new processing order π' is selected randomly among those from the neighborhood $N(\pi)$ of current processing order π. This processing order (solution) can provide either $C_{sum}(\pi') \leq C_{sum}(\pi)$ or $C_{sum}(\pi') > C_{sum}(\pi)$. In the former case π' is accepted immediately as the new solution for the next iteration, i.e. $\pi = \pi'$. In the latter case π' is accepted as the new solution with the probability $\exp(\Delta/T)$, where $\Delta = C_{sum}(\pi') - C_{sum}(\pi)$ - and T is a parameter called a temperature at iteration. The temperature T is getting changed along iterations by the use of a cooling scheme. A number of iterations, say m, is performed at the fixed temperature. Although the cooling should be carried out very slowly, most of the authors consider the change of the temperature at every iteration ($m = 1$). Two schemes of the temperature modification are commonly used: geometric $T_{i+1} = \lambda_i T_i$ and logarithmic $T_{i+1} = 1/(1 + \lambda_i T_i)$, $i = 1, \ldots, N$, where N is the total number of iterations, λ_i is a parameter, and T_0 is an initial temperature.

Aarts and van Laarhooven [1] have also made several suggestions concerning the choice of initial solution π^0, initial temperature T_0, λ_i, m and the stopping criterion. They have proposed to select π^0 at random, which helps with randomizing the search and removing solution dependence on π^0. The initial temperature is set to be $k = 10$ times the maximum value $\Delta_{\max} = max_{1 \leq i \leq k}(C_{sum}(\pi^i) - C_{sum}(\pi^{i-1}))$ between any two successive perturbed solutions when both are accepted, where π^i is the current solution in i-th iteration of algorithm. The initial temperature is set $T_0 = -\Delta_{\max}/\ln(p)$, where $p = 0.9$, the logarithmic cooling scheme parameter $\lambda_i = ln(1+\delta)/3\sigma_i$ where δ is the parameter of closeness to the equilibrium (0.1 - 10.0) and σ_i is the standard deviation of $C_{sum}(\pi)$ for all π generated at the temperature T. The value m equals the number (or its fraction) of different solutions that can be reached from the given one by introducing a single perturbation.

3.1 Neighborhood

The neighborhood $N(V,\pi)$ of a solution π is defined as a set of new solutions generated by the set of moves $V(\pi)$. The move $v \in V(\pi)$ transforms (perturbs) solution $\pi \in \Pi$ into another one $\pi^{(v)} \in \Pi$. One of the well-known transition operators for the job shop problem is the swap operator which takes two adjacent operation x and y and insert the operation y into the original position of operation x and x into the original position of operation y. The swap move can be unambiguously described by the pair of adjacent operation $v = (x, y)$. The set of moves $V(\pi)$ consists of all such moves that can be applied to π. For each machine $i \in M$ there are $n_k - 1$ possible swap moves. Thus, the size of this set and neighborhood is $o - m = \sum_{k=1}^{m}(n_k - 1)$. Unfortunately, $N(V,\pi)$ contains a huge number of unfeasible solutions as well as a quite large number of solutions worse than π.

Based on the Theorem 1 and Property 3 we can reduce the set $V(\pi)$ to the set $X(\pi) \subset V(\pi)$ which consists of only feasible and perspective moves. Formally, the set $X(\pi)$ is defined by the following formula

$$X(\pi) = \{(x,y) \in V(\pi) : \ x = u_i(j), \ y = u_i(j+1),$$
$$j = g_i^k, \ldots, h_i^k - 1, \ k = 1, \ldots, r_i, \ i \in O^L\}. \qquad (5)$$

The size of $X(\pi)$ strongly depends on the distribution of blocks in π.

3.2 The Representative Neighborhood

As already mentioned, in each iteration, the SA algorithm selects in $N(X,\pi)$ randomly a single solution neighboring to π. We consider also an alternative method of selecting neighbors which refers to the idea of representatives, see Nowicki and Smutnicki, [9], applied to the tabu search method for the permutation flow shop problem with a makespan criterion. This method has been also successively applied by Yamada and Reeves [12] for the permutation flow shop problem with the total completion time criterion.

In the *representative neighborhood* the large original neighborhood $N(X,\pi)$ is shared into small subsets (clusters). The representative of the cluster is the best solution in this cluster. Selection is made among representatives. Notice that selection of representatives requires significantly greater computational effort than for the conventional SA method. Hence, this method is useful for problems having effective *accelerators* and/or effective methods of parallel computing.

4 Parallel Computation of the Objective Function

In this section we propose the original method of parallel computation of C_{sum} criterion for the given set of neighbors $N(\pi)$ of the current solution π. At the begin, we will analyze properties of π and $\pi^{(v)}$, where v is a move. In the description we refer to the well known method of computing C_i, $i \in O$ values.

Fact 1. Values C_i, $i \in O$, can be found by using the recursive formula

$$C_i = \max(C_{\underline{s}_i}, C_{\underline{t}_i}) + p_i, \text{ where } C_0 = 0. \tag{6}$$

The application of (6) is correct if nodes of the graph are revised in a suitable order. Let $T_\pi = (t_1, \dots, t_o)$ be a topological order of nodes in graph $G(\pi)$. Note that T_π can be perceived as a permutation of elements from the set O.

Fact 2. The topological order T_π, for the fixed feasible π, can be found in the $O(o)$ time.

Fact 3. Values C_i, $i \in O$, for the fixed feasible π, can be found by running (6) for $i = t_1, \dots, t_o$. It requires the $O(o)$ time.

Using Facts (1)–(3) one can propose the following Procedure C of calculating C_i, $i \in O$, for the fixed π. The computational complexity of this procedure is $O(o)$.

PROCEDURE C

Step 1. Find the topological order T_π. If it does not exist return the *unfeasible solution.*

Step 2. Calculate values C_i, $i \in O$, by using (6) for $i = t_1, \dots, t_o$, where $(t_1, \dots, t_o) = T_\pi$.

Let us analyze the quick method of obtaining $T_{\pi^{(v)}}$ and C_i, $i \in O$, after the swap move $v = (x, y)$ made from π. Let T_π^{-1} be the inverse permutation to T_π. The element $T_\pi^{-1}(x)$ denotes the position of x in T_π. Clearly, we have $T_\pi^{-1}(x) < T_\pi^{-1}(y)$ for move $v = (x, y)$. It is easy to verify that $T_{\pi^{(v)}}$ can be obtained by reordering in T_π elements from position $T_\pi^{-1}(x)$ to position $T_\pi^{-1}(y)$. It takes $O(T_\pi^{-1}(y) - T_\pi^{-1}(x) + 1)$ time. For frequently met case $T_\pi^{-1}(y) = T_\pi^{-1}(x) + 1$ the computation complexity is $O(1)$. Unfortunately, regarding to C_i, the change of completion time of two swapped operations should be broadcasted to all successive operations; then updating of C_i from the position $T_\pi^{-1}(x)$ to position o, in $T_{\pi^{(v)}}$ obtained by reordering T_π, requires $O(o - T_\pi^{-1}(x) + 1)$ time.

Now, we are ready to present our parallel computing method dedicated to the fast calculation of objective function values for a set of neighbors. Among the known parallel computation models we select the vector calculations which are the most promising now and easy in hardware implementation. The selected model is the special case of the single instruction multiple data (SIMD) model.

Assume that the vector processor operates on vectors consisting of s elements. Let $V_s = \{v_1, \dots, v_s\}$ be a subset of parallel computed neighbors and let

$\overline{C}_i = (\overline{C}_i^1, \ldots, \overline{C}_i^s)$, where $\overline{C}_i^k$, $k = 1, \ldots, s$, denote completion times of operations $i \in O$ calculated for the k-th neighbor. In a similar way we define the vector of processing times $\overline{P}_i = (\overline{P}_i^1, \ldots, \overline{P}_i^s)$, obviously $\overline{P}_i^1 = \overline{P}_i^2 =, \ldots, = \overline{P}_i^s = p_i$. For each $v = (x, y) \in V_s$ we define three positions in T_π : (i) $f_{T_\pi}(v) = T_\pi^{-1}(x)$ the first position of updating T_π, (ii) $l_{T_\pi}(v) = T_\pi^{-1}(y) = T_\pi^{-1}(\overline{s}_x)$ the last position of updating T_π, (iii) $l_C(v)$ the last position of updating $\overline{C}^k$. The $l_C(v) = T_\pi^{-1}(y)$ if operation y is the last operation executed on the proper machines and $l_C(v) = T_\pi^{-1}(\overline{s}_y)$ in the opposite case. Finally, we reorder moves from the set V_s according to the non-decreasing value of $l_C(v)$. The proposed method is outlined in the Procedure P.

PROCEDURE P

```
Step 0: set $\overline{C}_0 = 0$.
Step 1: for $i = 1$ to $o$ do
Step 1.1: Calculate values $\overline{C}_{t_i}$ by using (6).
Step 1.2: for each $k$ such that $l_C(v_k) = i$ do
Step 1.2.1: execute move $v_k$ in $\pi$
Step 1.2.2: reorder $T_\pi$ from position $f_{T_\pi}(v_k)$ to $l_{T_\pi}(v_k)$
Step 1.2.3: calculate values $\overline{C}_{t_i}^k$ from position $f_{T_\pi}(v_k)$ to $l_C(v_k)$.
Step 1.2.4: restore $\pi$ and $T_\pi$
```

The initial Step 0 is clear. In the Step 1.1 all values of the vector $\overline{C}_{t_i}$ are calculated in parallel. This step is performed for $t_1, \ldots, t_o$ and takes $O(o)$ time. It is easy to verify that computations in Step 1.1 are performed according to order T_π determined by π. Therefore, for each π_v, $v \in V_s$ it is necessary to recalculate the completion times for all operations whose position have changed, i.e. from the position $f_{T_\pi}(v_k)$ to $l_{T_\pi}(v_k)$ and additionally for the successor of the operation y.

The computational complexity of the Step 1.2.1 is $O(1)$, for Step 1.2.2 and 1.2.4 is $O(l_{T_\pi}(v_k) - l_{T_\pi}(v_k))$. Step 1.2.3 is the most time consuming and takes $O(l_C(v_k) - l_{T_\pi}(v_k))$ time. The total time required by Step 1.2 is $O(\sum_{v \in V_s}(l_C(v) - f_{T_\pi}(v)))$. In the optimistic case, namely $l_C(v_k) - l_{T_\pi}(v_k) = 2$ for all $k = 1, ..., s$, the computational complexity of the algorithm is $O(o + s)$.

4.1 Multiple Mathematical Instructions

Contemporary used PCs are equipped with processors having the extended instruction MMX set. These special instructions allow one to make vectoring computations. The single instruction operates in a single processor cycle on extended registers (8 bytes). In the MMX set, it can be distinguished three groups of vector instructions operating on vector 8×1, 4×2, 2×4, where the former number denotes the vector size and the latter number – the data size. Since for the tested instances all performed values (C_i, p_i) can be coded on two bytes, we can perform vectoring operation on the vector of a size equals 4.

The seven main steps of parallel computation of expression (6) using MMX intrinsics are shown in Fig. 1. In the steps 1 and 2, the MMX registers mm0 and mm1 are loaded by data following from the vectors $\overline{C}_{s_i}$ and $\overline{C}_{t_i}$ respectively. The calculation of value $\max(\overline{C}_{s_i}, \overline{C}_{t_i})$ are decomposed into two steps: 3 and 4. At first, the value of $\max(0, \overline{C}_{s_i} - \overline{C}_{t_i})$ are calculated by using subtracts with saturation MMX intrinsic (step 3), the result is stored into the $mm0$ register. Afterwards the content of the $mm0$ register is increased by $\overline{C}_{t_i}$ (step 4). The values of vector $\overline{P}_i$ are stored in the $mm1$ register (step 5) and added to the contents of the $mm0$ register (step 6). The final results (contents of the $mm0$ register) are stored into memory in the step 7.

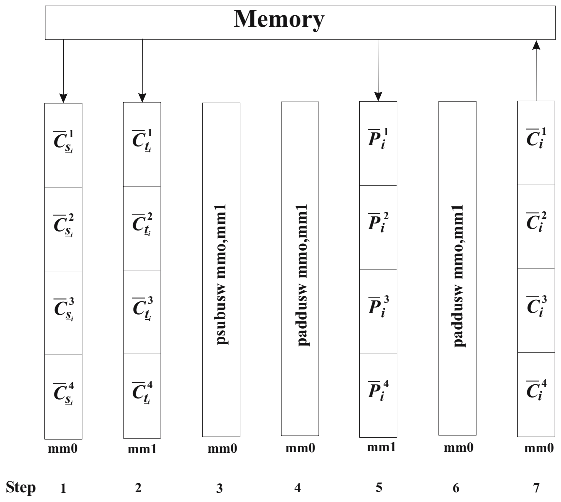


Fig. 1. Parallel computation of expression (6) for the given operation i

5 Computational Experiments

We have implemented three algorithms based on the simulated annealing method. In the first algorithm PSA-R we have used the representative-based neighborhood. From the neighborhood $N(X, \pi)$ we can select randomly $s = 4$ neighbors and we compute in parallel the value of the objective function. From the set obtained now we select the best solution. In the second PSA algorithm, for each solution $s = 4$ moves from $X(\pi)$ are generated at random. Moves are applied in turn to order π until a new solution is accepted. The objective function for all the solutions is computed in parallel. It is easy to observe that PSA emulates a traditional one-thread simulated annealing (SA) algorithm. The classic SA algorithm is the third from the implemented and tested algorithms.

Algorithms were coded in Visual C++ 2008 Express Edition, ran on a PC with Intel Core 2 Duo 2.66 GHz processor and the Windows XP operating system, and tested on 50 benchmark instances provided by Taillard [13]. The benchmark set contains 5 groups of hard instances in different sizes. For each size (group) $n \times m$: $15 \times 15, 20 \times 15, 20 \times 20, 30 \times 15, 30 \times 20$ a set of 10 instances was provided. In our tests all the values of tuning parameters for algorithms are found in the automatic way described in the previous section. Each algorithm was terminated after performed 1000 iterations. At the fixed temperature, the SA-R algorithm performs n iteration, whereas SA and PSA algorithms $4n$ iteration, i.e. all algorithms calculated the objective function value for the same number of solutions.

For each test instance and for each run of algorithm we have collected the following values: π^{ref} – reference solution – the best solution found in all runs of algorithms, $PRD = 100 \cdot (C_{sum}(\pi) - C_{sum}(\pi^{ref}))/C_{sum}(\pi^{ref})$ – the value of the percentage relative difference between C_{sum} function values for the solution π and reference solution π^{ref}, CPU – total computations time (in seconds). For each instance and for each algorithm based on 10 solutions generated during each of 10 runs we have calculated the following values: $MPRD$ – minimal PRD value, $APRD$ – average PRD value, $ACPU$ – average computation time (in seconds).

Table 1. Computational results of PSA-R, PSA and SA for PRD and CPU values

Group	PSA-R		PSA (SA)		ACPU time			Speedup ratio	
	MPRD	APRD	MPRD	APRD	PSAR	PSA	SA	PSAR	PSA
15×15	1.27	2.72	0.11	1.07	3.9	10.3	18.7	4.7	1.8
20×15	1.38	3.10	0.11	1.21	6.7	19.3	34.2	5.1	1.8
20×20	0.51	1.72	0.23	1.43	11.3	33.6	61.6	5.4	1.8
30×15	0.60	3.02	0.42	1.45	14.8	46.9	81.2	5.5	1.7
30×20	0.20	2.23	1.11	1.97	26.1	83.9	147.2	5.6	1.8

Table 1 shows results of computational experiments. The main observation is that the proposed method of SA algorithm parallelization significantly reduces the computation time. The speedup values are from 4.7 to 5.6 and increase with the increasing number of machines. It is easy to notice that the speedup is greater than the theoretical one (≤ 4) (we are obtaining superlinear speedup). The additional speedup is obtained due to the MMX instruction utilization, which eliminates branches in the executing code. The branches are essential for implementation of max function in x86 set of instruction. Comparing computations time of PSA and SA one can observe that the speedup is significantly smaller (1.8). With respect to PRD values it has been pointed that PSA provides better results than PSA-R for the majority of groups of instance. For the first five groups the MPRD values are from 0.5 to 1.4 for PSA-R, whereas from 0.1 to 0.4 for PSA (SA). In the last group of instances, conversely PSA-R provides better results (MPRD=0.2) than PSA-R (MPRD=0.2). It can be notice that for this group of instances we observe the highest speedup value.

6 Conclusions

To the best of our knowledge, this is the first paper which deals with the small-grain parallelization of algorithms for the job shop problem. A special attention has been paid to the kind of parallelism which can be easily applied to the new generation of processors installed in PCs, with multiple cores (dual, quad, etc.) as well as with the extended set of instructions (such as MMX and SSE2). The proposed methods have been applied successfully to various simulated annealing metaheuristics for the job shop problem with C_{sum} criterion.

References

1. Aarts, E.H.L., van Laarhoven, P.J.M.: Simulated annealing: a pedestrain review of the theory and some aplications. In: Deviijver, P.A., Kittler, J. (eds.) Pattern Recognition and Applications. Springer, Berlin (1987)
2. Armentano, V.A., Scrich, C.R.: Tabu search for minimizing total tardiness in a job shop. International Journal of Production Economics 63(2), 131–140 (2000)
3. Bushee, D.C., Svestka, J.A.: A bi-directional scheduling approach for job shops. International Journal of Production Research 37(16), 3823–3837 (1999)
4. Grabowski, J., Wodecki, M.: A very fast tabu search algorithm for job shop problem. In: Rego, C., Alidaee, B. (eds.) Metaheuristic optimization via memory and evolution. Tabu search and scatter search, vol. 30, pp. 117–144. Kluwer Academic Publ., Boston (2005)
5. He, Z., Yang, T., Tiger, A.: An exchange heuristic embedded with simulated annealing for due-dates job-shop scheduling. European Journal of Operational Research 91, 99–117 (1996)
6. Holthaus, O., Rajendran, C.: Efficient jobshop dispatching rules: further developments. Production Planning and Control 11, 171–178 (2000)
7. Jain, A.S., Rangaswamy, B., Meeran, S.: New and stronger job-shop neighborhoods: A focus on the method of Nowicki and Smutnicki (1996). Journal of Heuristics 6(4), 457–480 (2000)
8. Mattfeld, D.C., Bierwirth, C.: An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 155(3), 616–630 (2004)
9. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research 19(1), 160–175 (1996)
10. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8, 145–159 (2005)
11. Pezzella, F., Merelli, E.: A tabu search method guided by shifting bottleneck for the job-shop scheduling problem. European Journal of Operational Research 120, 297–310 (2000)
12. Reeves, C.R., Yamada, T.: Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search ICEC 1998. In: IEEE International Conference on Evolutionary Computation, pp. 230–234 (1998)
13. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
14. Wang, T.Y., Wu, K.B.: An eficient configuration generation mechanism to solve job shop scheduling problems by the simulated annealing. International Journal of Systems Science 30(5), 527–532 (1999)

Developing Scientific Applications with Loosely-Coupled Sub-tasks

Shantenu Jha, Yaakoub El-Khamra, and Joohyun Kim

Center for Computation and Technology, Louisiana State University, Baton Rouge LA 70803, USA

Abstract. The Simple API for Grid Applications (SAGA) can be used to develop a range of applications which are in turn composed of multiple sub-tasks. In particular SAGA is an effective tool for coordinating and orchestrating the many sub-tasks of such applications, whilst keeping the application agnostic to the details of the infrastructure used. Although developed primarily in the context of distributed applications, SAGA provides an equally valid approach for applications with many sub-tasks on single high-end supercomputers, such as emerging peta-scale computers. Specifically, in this paper we describe how SAGA has been used to develop applications from two types of applications: the first with loosely-coupled homogeneous sub-tasks and, applications with loosely-coupled heterogeneous sub-tasks. We also analyse and contrast the coupling and scheduling requirements of the sub-tasks for these two applications. We find that applications with multiple sub-tasks often have dynamic characteristics, and thus require support for both infrastructure-independent programming models and agile execution models. Hence attention must be paid to the practical deployment challenges along with the theoretical advances in the development of infrastructure-independent applications.

1 Introduction

There exist many scientific problems that are solved by the collective analysis of many independent tasks, e.g., Monte-Carlo simulations, or parameter sweeps. There also exists a large class of scientific problems that involve applications that can either be decomposed into smaller coupled sub-tasks via the of choice an appropriate algorithm [1], or are naturally composed of coupled sub-tasks. The decomposition of an otherwise monolithic application into smaller components of computation, in principle makes them amenable to efficient distribution.

In this paper, we discuss Replica-Exchange (RE) and Ensemble Kalman-Filter (EnKF) based applications, as representative prototypes of applications with coupled sub-tasks. Although similar at some levels, they possess important differences. For RE based simulations, the sub-tasks are identical (ie. replicas), whereas for the EnKF the sub-tasks are heterogeneous. Additionally the nature of coupling between the sub-tasks in the former (regular intervals and pair-wise) is very different from the latter (irregular and a global-synchronisation point).

It is important to appreciate the difference between loosely-coupled when typically used in the context of parallel applications versus when used in the context of distributed applications. For the former, loosely-coupling is most often a reference to the application's tolerance of latency in message passing. For distributed applications loose (or tight) coupling has more context: it could be a reference to the flexibility in scheduling and placing the sub-tasks or even a flexibility in choice of resources the different sub-tasks are mapped to. Although both applications we investigate are classified as loosely-coupled, the nature of the coupling between their sub-tasks varies. It is important to appreciate The nature of the coupling of the sub-tasks, in addition to imposing constraints on scheduling and resource mapping strategies, also determines the feasibility of any speculative computing. Thus, along with the size and number of sub-tasks, the nature of coupling determines the overall development and deployment strategy.

In addition to some similarity between application characteristics, what binds the two together, is our adopted approach of developing distributed applications. The Simple API for Grid Applications (SAGA) [2] provides a simple, standard, programmatic approach to codify distributed applications such that they can be seamlessly run on any underlying infrastructure. Critically, this allows the application developer to focus on supporting the application characteristics and exploiting the relative strengths of different infrastructure whilst not worrying about adapting to the details of the infrastructure. Not being coupled to the details of the underlying infrastructure is a necessary condition for applications whose resource requirement might increase or those that want to make opportunistic use of newly available resources. In other words independence from specific infrastructure, is a necessary condition for dynamic applications to achieve the desired agile-execution models and thus be adaptive. The aim of this paper is to discuss how SAGA has been used to develop two applications with multiple sub-tasks in a way such that these applications can be deployed and executed on both distributed as well high-end machines, with a minimal, if not no-changes.

Lingering problems associated with deployment on production Grids has made the uptake of Grids challenging and unattractive to the end-scientist. In a nutshell, our experience is consistent with and indicates that one of the reasons deployment on general-purpose Grids is difficult, because Grids are comprised of many "isolated" components. We believe that this contributes to a currently unmanageable number degrees-of-freedom and failure modes. Although programming models and conceptual frameworks exists to unify the uptake of "grids or supercomputers" as required, practical considerations make this currently unrealistic and motivate the end-scientist to settle for the the solution that is often simpler to deploy. Whereas this has consequences for all distributed applications, it influences the development and uptake of *dynamic* distributed applications. Although the focus here is on utilising distributed machines, the same approach can be used for monolithic large machines. We demonstrate the validity of the SAGA approach for high-performance computing on large single machines.

2 SAGA: A Standard Programming Interface

The Simple API for Grid Applications (SAGA) is an API standardization effort within the Open Grid Forum (OGF) [3] an international standards development

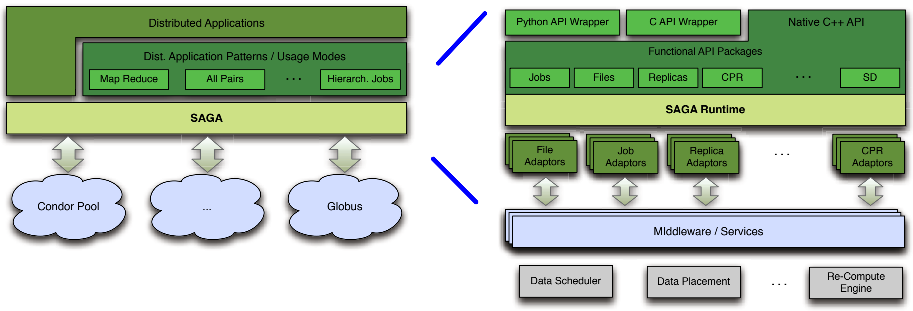


Fig. 1. Schematic diagram showing how SAGA supports the development of three simple, but important ways of developing distributed applications. Layered schematic of the different components of the SAGA landscape. The core API supports the main functionality required by distributed applications. Middleware specific adaptors make applications developed using SAGA grid portable.

body concerned primarily with standards for distributed computing. SAGA provides a simple, POSIX-style API to the most common Grid functions at a sufficiently high-level of abstraction so as to be able to be independent of the diverse and dynamic Grid environments. The SAGA specification defines interfaces for the most common Grid-programming functions grouped as a set of functional packages. The SAGA Version 1.0 specification defines the following packages:

- File package - provides methods for accessing local and remote filesystems, browsing directories, moving, copying, and deleting files, setting access permissions, as well as zero-copy reading and writing. The replica package support the same functionality for logical files.
- Job package - provides methods for describing, submitting, monitoring, and controlling local and remote jobs.
- Stream package - provides methods for authenticated local and remote socket connections with hooks to support authorization and encryption schemes.
- RPC package - is an implementation of the OGF GridRPC API [4] definition and provides methods for unified remote procedure calls.

The SAGA Runtime Engine can dynamically load environment specific adaptor (see Fig. 1). The two critical aspects of SAGA are its *simplicity* of use and the fact that it is a proposed standard. It is important to note, that these two properties provide the added value of using SAGA for distributed application development. Simplicity arises from being able to limit the scope to only the most common and important grid-functionality required by applications. Standardization represents the fact that the interface is derived from a wide-range of applications using a collaborative approach and the output of which is endorsed by the broader community.

2.1 Developing Distributed Applications with SAGA

SAGA can be used to develop and support distributed applications in many different ways; the exact way in which it is used, in addition to the application characteristics, depends upon factors such as how the application needs to be used. For simplicity, in this paper, we will discuss only three different approaches for distributed application development (schematically summarized on the left side of Fig. 1). First, applications can use SAGA directly for standardised and simple distributed function calls that work on nearly all middleware systems. Typically, applications developed using direct SAGA calls are *explicitly* distributed. Secondly, SAGA can be used to create infrastructure independent frameworks (that support patterns such as MapReduce), which provide distributed capability and which can be used by applications to be *implicitly* distributed. Thirdly, SAGA can be used to support usage modes that provide access to distributed infrastructure, such as bulk job-submission or hierarchical job-submission over different machines. For this case too, applications are typically implicitly distributed, and the knowledge/control of utilizing distributed infrastructure is left to the SAGA-based framework that supports the usage-mode. The RE application that we will discuss in the paper belongs to the third category, whilst the EnKF based application is of the first type.

3 Applications with Loosely-Coupled Homogeneous Sub-tasks: Replica-Exchange

RE [5] simulations can be used to understand important physical phenomena – ranging from protein folding dynamics to binding affinity calculations required for computational drug discovery. For RE simulations utilizing as many distributed resources as possible, is critical for the effective solution of the scientific problem [6]. Distributed RE simulations must be able to orchestrate different resources in a complex and dynamic environment. Writing such an applications is a complex task for a myriad number of reasons [7]. In the following a SAGA-based RE framework developed for molecular dynamics simulations is described.

3.1 Application Description

Even with the most powerful computing resources at the moment, straightforward Molecular Dynamics (MD) simulations are unable to reach the relevant time-scales required to study conformational changes and searches. This is partly due to the inherent limitations in the MD algorithm – a global synchronization is required at the end of each time step. This limitation provides an important motivation for research into finding ways to accelerate sampling and enhance "effective" time-scales studied. Generalized ensemble approaches – of which Replica-Exchange Molecular Dynamics (REMD) [5] are a prominent example – represent an important and promising attempt to overcome the general limitations of insufficient time-scales, as well as specific limitations of inadequate conformational sampling arising from kinetic trappings. In the simplest formulation, RE is an algorithm whereby one single long-running simulation is be substituted for an ensemble of shorter-running similar simulations, but which are very loosely-coupled, ie, the interval between exchange attempts is much larger than the interval over which the simulations run; this also make the RE formulation of physical problems excellent candidates for distributed environments.

3.2 Application Architecture

There are many architectural aspects of the framework used to implement RE simulations. However, we will focus on the abstractions that we create using SAGA that enable efficient job-submission on any underlying infrastructure. Details of the architecture and abstractions can be found in Ref. [6,7]. Here we present the architecture in the context of an application consisting of loosely-coupled multiple sub-tasks. RE simulations can be thought of as consisting of two distinct components: the simulation engine/mechanism used for each replica process, and the orchestration-coupling mechanism between the individual replicas. Our current RE framework uses NAMD for the former and a SAGA-based framework for orchestration and coordination of the replica sub-tasks.

The developed RE framework [7] comprises of the *RE-Manager* – the central master deployed on the user's desktop, and the *Replica-Agents*, that reside on the machines where RE simulations are carried out. The RE-Manager orchestrates all replicas, i. e. it is responsible for the parameterization of replica tasks, file staging, job spawning and the conduction of the replica-exchange itself. The Replica-Agent is responsible for spawning and monitoring the sub-tasks.

In particular, queueing delays can represent a major bottleneck: a single crowded resource can slowdown the simulation arbitrary. Thus, to achieve an optimal time to solution, RE sub-tasks need to be dispatched efficiently. A common principle to prevent this is the usage of Glide-In jobs, which represent a placeholder for a set of sub-tasks (see Ref. [8]). For a Glide-In job, a sufficiently large chunk of resources is requested. Smaller sub-tasks can then rapidly be executed through the Glide-In job. Figure 2 summarizes the abstractions used within the RE framework.

While the implementation of the enhanced job model is entirely based on SAGA we can utilise other frameworks, such as the original Condor Glide-In [8]. Currently, we are actively working on a Condor adaptor for SAGA [9], which will also support native Glide-In functionality for Condor Jobs; our enhanced job model will then serve as abstraction, while the Condor level Glide-In will be used where appropriate. Irrespective of that, however, the strengths of our approach are the following: A general purpose Glide-in mechanism that does not require either Condor, or Globus and in which sub-tasks are part of a Glide-In meta-job, can be controlled at the application-level using simple *ssh* if needed. Secondly, the same mechanisms can be used to exploit distributed resources [6], as well as single high-end resources, without any changes in application code. This represents the basis of our claim of independence from underlying-infrastructure.

3.3 Application Deployment

We have shown how using the SAGA Glide-In infrastructure on multiple TeraGrid/LONI resources, the time-to-solution can be reduced [10]. Continuing with

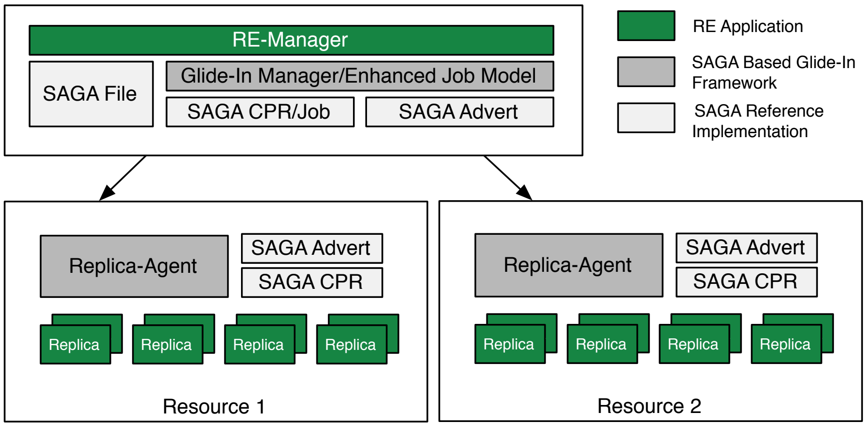


Fig. 2. Replica Exchange Framework Abstractions: The Replica-Agent is used as place-holder job for all sub-tasks running on a single cluster. The RE-Manager can control both the Replica-Agents and the replica jobs using a SAGA-based user-level job API. By using this efficient way to allocate resources, queuing times are minimized and the time to completion can be dramatically reduced when using multiple and single resources.

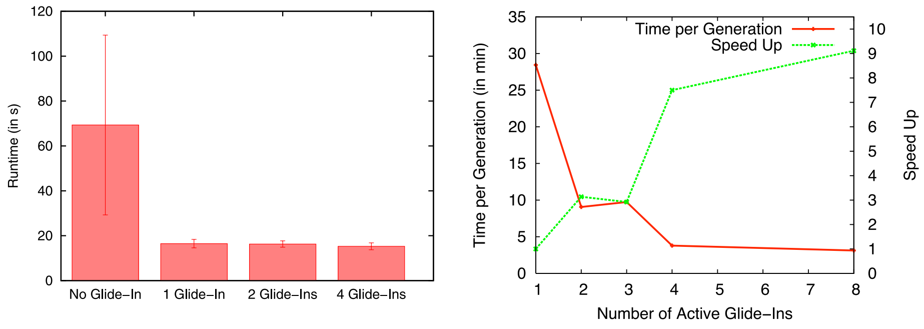


Fig. 3. SAGA Glide-In Performance: The figure shows the average runtime of a RE simulation with 16 RE processes running on 16 cores each on QueenBee. The Glide-In framework provides the possibility to effectively cluster RE jobs to receive a significant reduced time to solution (upto 80 %) even on a single machine. The plot on the right shows the number of active Glide-Ins over a six-hour run on the TeraGrid. The plot in red (using the left-hand y axis) illustrates how the average time between exchange attempts (inverse of physical efficiency) decreases as the number of Glide-Ins increases. The plot in green shows the speedup.

the theme that well developed abstractions can serve across the spectrum – distributed HPC machines (such as the TeraGrid) to single high-end supercomputers (such as Abe or QueenBee) to many smaller machines flocked together (Condor style high-throughput), we focus on using the same infrastructure to reduce the time-to-solution on a single machine. This is also a test of the scalability of the SAGA-based Glide-In framework on a single machines.

Figure 3(a) shows that the Glide-In framework is especially beneficial if there are fluctuations in the queue-time for the sub-tasks (which is almost always!). The more sub-tasks are spawned, the more likely such delays become. While with the SAGA Glide-In framework the runtime only modestly increase with more than 8 replicas, the runtime rapidly rises when using regular Globus job for spawning NAMD tasks. The unpredictable nature of these queueing times becomes obvious by the high standard deviation found in the measurements.

Figure 3(a) shows that the SAGA Glide-In framework can provide a reduced time to solution even on a single machine by avoiding queuing time delays and fluctuations for every sub-task and allowing the efficient dispatching of RE tasks solely through the Replica-Agent. During our experiments we were able to measure speedups of up to 80 % compared to the non Glide-In approach. Fig. 3(b) shows several measures of how the use of SAGA framework results in efficient and effective deployment.

4 Loosely-Coupled Heterogeneous Sub-tasks: Kalman-Filter Simulations

Ensemble Kalman filters (EnKF) are widely used in science and engineering [11]. EnKF are recursive filters that can be used to handle large, noisy data. The data can be the set of results and parameters of ensembles of different models of a particular physical system. The ensembles are run through the KF to obtain the true physical state of the data [11], to effectively solve the inverse problem.

4.1 Application Description

In EnKF, an ensemble of forward models are run with different parameters. The data they produce is assimilated at the end of each stage, the parameters are corrected, and the models are run again. This process is repeated several times until a pre-determined criteria has been met. The ensemble of forward models are run as sub-tasks on possibly different machines, launched by a master filter task using SAGA. SAGA is also used to control the flow of data between the filter and the ensemble of models.

The variation in model parameters often has a direct and sizable influence on the complexity of solving the underlying equations, thus varying the required runtime of different models. Since we need both parameters and results for the EnKF, a mechanism to assign models to available resources based on their expected time to completion and resource requirement is useful. Such a mechanism estimates the time a model will spend in the queue of a resource, the time it needs to run, and the time required to migrate the data it requires/produces back and forth, and based on that attempt to minimize the time required to perform each Kalman filter iteration. In fact, with changing resource simulation requirements (as is the case with models that find themselves lagging behind the rest of the model pack), a mechanism which can take advantage of faster, cheaper or more powerful machines is even more advantageous [12].

We have developed a mechanism whereby EnKF can be solved using multiple-resources, using application-level scheduling applied dynamically [13], ie mapping the sub-tasks requirement to the resources available at the instant the sub-tasks become available and ready to run, as opposed to *a priori* static method of job submission. For the problem size studied, the sub-tasks required mostly less than 32 processors. For this paper we used the earlier developed frameworks and deployed it on a single large machine – NCSA's Abe [1]. A mechanism (multiple, distributed versus single machine) that is more efficient for physical models with sub-tasks that have typically low processor counts, will not necessarily be the more efficient as the typical sub-task size increases. Therefore it is crucial that any general-purpose solution be usable on both single large machines to multiple machines. We can enhance throughput further by applying the Glide-In mechanisms discussed in the earlier section, which facilitate dynamic tasks being aggregated from similar sub-tasks. We will report on the results of this and whether the framework can be used on high-end petascale supercomputers in future work.

While concurrently running on various machines is advantageous by simple virtue of the fact that more resources would be available for running the forward models, it is also more technically challenging than running on a single machine. Authentication, job launching, multiple executables in correct paths for different architectures and file systems, and of course file transfer across the different machines are all possible points of failure. These are just some of the additional reasons why a high-level interface such as SAGA is required to hide the heterogeneity of different distributed systems. In spite of that, several challenges remain – technical, sociological as well as policy level, some specific examples of relevance we discuss in the next section.

5 Deploying on Distributed Resources

As mentioned in the opening section, using SAGA we have developed programming and execution models, whereby applications are independent of the underlying infrastructure, i.e., either use a monolithic mammoth machine, or multiple-distributed machines, depending upon the physical problem being investigated, without any modification at the application-level code. In spite of these theoretical advances, in practise end-users often find it more convenient to use single resources, even if not optimally-efficient. The smooth and effective deployment of distributed applications on heterogeneous resources remains is a difficult task. To highlight just some of the challenges of deploying advanced application on general-purpose distributed infrastructure, we mention the fact that at best 33%

[1] We wanted to use Ranger, but BQP was not available on Ranger, and would not have been before the submission of this paper.

of the resources we tried were usable (ie two in three were not usable) . We mention two problems that we encountered and led to a high amount of complexity: different library versions and broken Globus installations. Also, Globus installations on TeraGrid machines proved to be quite different. For example, the Globus GRAM2 versions on Abe and QueenBee map the RSL count element different: While on QB the count element is mapped to the number of cores, on Abe this element describes the number of nodes. Further standardization of this aspect is required in the future. The GRAM2 on Ranger (in particular the Sun GridEngine adaptor) was completely unusable due to the lack of support for MPI jobs.

Deploying our applications on ranger was not a straightforward task. SAGA requires a recent installation of the BOOST library which we had to compile ourselves. When we were finished compiling our applications, we ran into a job submission problem on a particular login node. Moving past the firewall and GRAM2 issues, getting the right certificates that are recognized on the machine, we discovered there were even more issues that need to be resolved: GridFTP was not working, the Globus/SGE script had a small error in it that had to be corrected. These issues are outlined in tickets 4957, 5111, 5130, 5145, 5172 and 5174. It is important to note that we encountered excellent response time and expert system administrators who resolved all of these issues promptly, but reiterates the complexity of utilising multiple resources from general-purpose grids. The aim here is not to criticise any provider – resource or software product, but to simply highlight the practical challenges of deploying distributed applications.

6 Conclusions and Discussions

SAGA provides the abstractions and the ability to create applications with multiple sub-tasks that can exploit multiple and different infrastructure types. We have demonstrated this via the implementation of two distinct, specific applications but both representative of a broader class of applications and running them in two different execution environments without changing the application in any way! The specific applications differed not only in the types of sub-tasks (homogeneous versus heterogeneous) but also in the nature of the coupling between the sub-tasks.

It is interesting to note that *simple, naive* implementations of these applications are possible; these would require these applications to be "grid-unaware" (or implicitly distributed). Although we don't provide details here, the real power of these applications arise from their ability to have an agile execution model, i.e., by being adaptive to dynamic resource requirements or availability. In other words, in order to develop applications that have agile execution models, more often than not, applications need to explicitly control the distributed aspects, i.e., be *grid-aware* We posit that SAGA provides an important mechanism to develop explicitly distributed applications.

Optimal scheduling of sub-tasks remains a challenge of distributed computing; however as demonstrated, for adaptive applications, scheduling can often be done effectively at the application level. This is possible because, as shown, adaptive applications don't necessarily need tight co-scheduling, but often just lightweight-coupling between resources. This is yet another advantage of an agile-execution model.

Acknowledgement

Important funding for SAGA specification and development has been provided by the UK EPSRC grant number GR/D0766171/1. SJ also acknowledges the e-Science Institute, Edinburgh for supporting the research theme, "Distributed Programming Abstractions". This work would not have been possible without the efforts and support of the wider SAGA team. This work has also been made possible thanks to the internal resources of the Center for Computation & Technology (CCT) at Louisiana State University and computer resources provided by LONI.

References

1. Jha, S., Coveney, P., Harvey, M.: Spice: Simulated pore interactive computing environment. In: SC 2005: Proceedings of the ACM/IEEE conference on Supercomputing, p. 70. IEEE Computer Society, Los Alamitos (2005)
2. SAGA, http://saga.cct.lsu.edu
3. Open Grid Forum, http://www.ogf.org/
4. GridRPC, http://forge.ogf.org/sf/projects/gridrpc-wg
5. Sugita, Y., Okamoto, Y.: Replica-Exchange Molecular Dynamics Method for Protein Folding. Chemical Physics Letters 314, 141–151 (1999)
6. Luckow, A., Jha, S., Kim, J., Merzky, A., Schnor, B.: Adaptive Replica-Exchange Simulations. Royal Society Philosophical Transactions A (to appear, 2009)
7. Luckow, A., Jha, S., Kim, J., Merzky, A., Schnor, B.: Distributed replica-exchange simulations on production environments using saga and migol. Accepted for 4th IEEE International Conference on e-Science (2008)
8. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing 5(3), 237–246 (2002)
9. SAGA/Condor, http://fortytwo.cct.lsu.edu:8000/SAGA/wiki/CondorAdaptor
10. Luckow, A., Jha, S., Kim, J., Merzky, A., Schnor, B.: Distributed replica-exchange simulations on production environments using saga and migol. In: 4th IEEE International Conference on e-Science, Indianapolis, IN, USA (2008)
11. Kalman, R.E.: A new approach to linear filtering and prediction problems
12. Jha, S., Kaiser, H., Khamra, Y.E., Weidner, O.: Design and Implementation of Network Performance Aware Applications Using SAGA and Cactus. In: E-SCIENCE 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, pp. 143–150 (2007)
13. Jha, S., Khamra, Y.E., Kaiser, H., Merzky, A., Weidner, O.: Developing large-scale adaptive scientific applications with hard to predict runtime resource requirements. In: Proceedings of TeraGrid 2008 (2008), http://tinyurl.com/5du32j

Simulation of Multiphysics Multiscale Systems, 6th International Workshop

Simulation of Multiphysics Multiscale Systems, 6th International Workshop

Valeria V. Krzhizhanovskaya[1,2]

[1] Section Computational Science, Faculty of Science, University of Amsterdam, Science Park 107, 1098 XG Amsterdam, The Netherlands
`V.Krzhizhanovskaya@uva.nl`
`http://www.science.uva.nl/~valeria/SMMS`
[2] St. Petersburg State Polytechnic University, Russia

Abstract. Modeling and *Simulation of Multiphysics Multiscale Systems* (SMMS) poses a grand challenge to computational science. To adequately simulate numerous intertwined processes characterized by different spatial and temporal scales spanning many orders of magnitude, sophisticated models and advanced computational techniques are required. The aim of the SMMS workshop is to encourage and review the progress in this multidisciplinary research field. This short paper describes the scope of the workshop and gives pointers to the papers reflecting the latest developments in the field.

Keywords: Multiphysics, Multiscale, Complex systems, Modeling, Simulation, ICCS, SMMS, Workshop.

1 Introduction to the Workshop

The progress in understanding physical, chemical, biological, sociological and economical processes strongly depends on adequacy and accuracy of numerical simulation. All the systems important for scientific and industrial applications are inherently multiphysics and multiscale: they involve interactions amongst a wide range of physical phenomena operating at different spatial and temporal scales. Complex flows, fluid-structure interactions, plasma and chemical processes, thermo-mechanical and electromagnetic systems are just a few examples essential for fundamental and applied sciences. Numerical simulation of these multiphysics and multiscale problems requires development of sophisticated models and methods for their integration, as well as efficient numerical algorithms and advanced computational techniques.

To boost scientific cross-fertilization and promote collaboration of the diverse groups of specialists involved, we have launched a series of mini-symposia on Simulation of Multiphysics Multiscale Systems (SMMS) in conjunction with the International Conference on Computational Sciences (ICCS) [1].

The sixth workshop in this series, organized as a part of ICCS-2009, expands the scope of the meeting from physics and engineering to biological and biomedical applications. This includes computational models of tissue- and organo-genesis, tumor growth, blood vessel formation and interaction with the hosting tissue, biochemical transport and signaling, biomedical simulations for surgical planning, etc. The topics traditionally addressed by the symposium include modeling of multiphysics and/or multiscale systems on different levels of description, novel approaches to combine different models and scales in one problem solution, advanced numerical methods for solving multiphysics multiscale problems, new algorithms for parallel distributed computing specific to the field, and challenging multiphysics multiscale applications from industry and academia.

A large collection of rigorously reviewed papers selected for the workshops highlight modern trends and recent achievements [2]. It shows the progress made in coupling different models (such as continuous and discrete models; quantum and classical approaches; deterministic and stochastic techniques; nano, micro, meso and macro descriptions) and suggests various coupling approaches (e.g. homogenization techniques, multigrid and nested grids methods, variational multiscale methods; embedded, concurrent, integrated or hand-shaking multiscale methods, domain bridging methods, etc.). A number of selected papers have been published in the special issues of the International Journal for Multiscale Computational Engineering [3], collecting state-of-the-art methods for multiscale multiphysics applications.

Acknowledgments. We would like to thank the participants of our workshop for their inspiring contributions, and the members of the workshop program committee for their diligent work, which led to the very high quality of the conference. The organization of this event was partly supported by the Virtual Laboratory for e-Science Bsik project.

References

1. Simulation of Multiphysics Multiscale Systems, http://www.science.uva.nl/~valeria/SMMS
2. LNCS this volume, 16 papers after this introduction;
   LNCS V. 4487/2007. DOI: http://dx.doi.org/10.1007/978-3-540-69387-1 , pp. 165-340;
   LNCS V. 4487/2007. DOI: http://dx.doi.org/10.1007/978-3-540-72584-8 , pp. 755-954;
   LNCS V. 3992/2006. DOI: http://dx.doi.org/10.1007/11758525 , pp. 1-138;
   LNCS V. 3516/2005. DOI: http://dx.doi.org/10.1007/b136575 , pp. 1-146;
   LNCS V. 3039/2004. DOI: http://dx.doi.org/10.1007/b98005 , pp. 540-678.
3. Simulation of Multiphysics Multiscale Systems. Special Issues of the International Journal for Multiscale Computational Engineering:
   V. 4, Issue 2, 2006. DOI: http://dx.doi.org/10.1615/IntJMultCompEng.v4.i2;
   V. 4, Issue 3, 2006. DOI: http://dx.doi.org/10.1615/IntJMultCompEng.v4.i3;
   V. 5, Issue 1, 2007. DOI: http://dx.doi.org/10.1615/IntJMultCompEng.v5.i1;
   V. 6, Issue 1, 2008. DOI: http://dx.doi.org/10.1615/IntJMultCompEng.v6.i1;
   V. 7, 2009 – in preparation.

Two-Dimensional Micro-Hartmann Gas Flows

Chunpei Cai and Khaleel R.A. Khasawneh

Department of Mechanical & Aerospace Engineering
New Mexico State University, Las Cruces, New Mexico, 88003-8001, U.S.A.
ccai@nmsu.edu

Abstract. We analyze and simulate a near continuum MagnetoGas-Dynamic(MGD) flow inside a two-dimensional microchannel with a low magnetic Reynolds number assumption. Complex physics such as rarefication, electric and magnetic effects are considered in the asymptotic solutions. This work represents an extension from the classical Hartmann flow in a two-dimensional channel of infinite length to a microchannel of finite length. We obtain a non-dimensional equation that relates the pressure ratio, Reynolds number, Mach number, magnetic Reynolds number and magnetic force number. We also solved for asymptotic solutions of compressible gas flow based on the velocity-slip and temperature-jump wall boundary conditions while maintaining a consistent quasi-isothermal assumption. Numerical solutions of the same formulation are obtained for validation of the present analytical solutions.

Keywords: multi-scales, micro-flows, rarefication flows, Hartmann flows.

1 Introduction

Microchannels are important components for many Micro-Electro-Mechanical Systems(MEMS).The study of gaseous flows inside microchannels has been an interesting research topic. There are many reports in the literature about gas flows in microchannels and micro-tubes. [1] A microchannel for fuel cells, for example, is of dimensions of micrometers, whereas a channel of conventional dimensions is of order of centimeters to meters. It is well accepted that for near continuum gas flows through microchannels, the Navier-Stokes equations are valid if a slip wall boundary condition is used. Many researchers have obtained theoretical solutions for the flow distributions along a microchannel with an isothermal assumption, including the pioneering work by Arkilic *et al*,[1] and Zohar *et al.*[2] Numerically, there are many methods for simulating compressible flows in a microchannel: with the direct simulation Monte Carlo method, the Information Preservation method,[3] gas-kinetic BGK-Burnett equation solutions.[4] Discussions of thermal heating effects are reported as well.[5]

One important problem related to gas flow inside a microchannel is conductive gas flows under external magnetic and electric fields, or MagnetoGasDynamic(MGD) flows. These fields can significantly affect the internal gas flow field, while different placements of electric field enables the channel to perform as a generator, a pump, a flux meter, or an accelerator.[6] This work is an extension from the classical Hartmann flow[6] to a flow in a microchannel of finite length with the follow differences. First, for pressure driven gas flows inside a microchannel, usually there are large density changes inside the microchannel, and the average velocity can not remain unchanged in order to maintain a constant mass flow rate. Second, the boundary conditions are different. For Hartmann flow, there is essentially no variation in the flow direction; wall boundary conditions are nonslip and constant temperature; one can use periodic boundary conditions (except for pressure) at the inlet and outlet.[7] However, for microchannel flows with density variations and rarefication effects, we will not use periodic boundary conditions at inlet and outlet; further, we will specify a set of general velocity-slip and temperature-jump wall boundary conditions, which include the continuum no-slip and temperature boundary conditions as a special case. Third, the pressure gradient inside a microchannel of finite length is not assumed constant throughout. Simply taking a linear pressure distribution assumption (after the Hartmann flow) for the present microchannel flow will result in an inaccurate solution.

The work in this paper is a natural extension from two previous work by Arkilic *et al* [1] and Cai *et al.*[5]Along the same vein, the present study follows Cai's previous approach [5] for treatments of microchannel MGD flows. A few works appeared in recent years in the same area of MGD microchannel flow.[8,9] There are some differences between the present approach and others: 1. Our study includes a governing equation for temperature necessitated by the Joule heating effects to obtain the temperature field;[5] by comparison, the past work completely neglects the energy equation. 2. We utilize the low magnetic Reynolds number assumption, which is crucial to simplify the MGD equations; we have not noticed any previous treatment for the same problem in the literature. Without this assumption, the magnetic field variations in general should be considered; thus it is unlikely that such asymptotic approach can be consistently formulated. 3. According to the X-momentum equation, we provided the governing relation between the Re and Ma numbers and two other nondimensional parameters for magnetic and electric fields. We then conducted a consistent order of magnitude analysis. 4. We obtain a full set of asymptotic solutions in a microchannel flow. The nonlinear pressure solution implies that without it the solution for U-Velocity is incorrect.

2 Problem Description and Order Estimations

As illustrated by Figure 1, a microchannel has a height of $2d$ and a length of L, and the average compressible gas properties of pressure, density, velocity and number density are p_o, ρ_o, U_o, n_o at the channel outlet. The averaged outlet quantities are adopted to normalize the following governing equations and boundary conditions. The inlet pressure is several times larger than the outlet pressure, their pressure ratio is denoted as P. The coordinate origin is set at the inlet center point, and the X-axis is along the channel centerline, the Y-axis is along the direction normal to the channel wall. Further we assume:

1. External magnetic field is fixed at a value $(0, B_0, 0)$, along the Y-direction only, electric field is along the Z direction only $(0, 0, E_z)$.
2. The flow is well approximated with a quasi-isothermal assumption.[5] As such, gas viscosity, μ, thermal conductivity, k, magnetic permeability, μ_m, and electric conductivity, σ, are treated as constants.
3. The magnetic Reynolds number, $R_\sigma = (2d)U_o\mu_o\sigma$, is very small. It renders a negligible induced magnetic field, B_x compared with B_0.
4. The flow is two-dimensional, hence $\partial/\partial z = 0$, $w = 0$.
5. We assume the electric field is linked with the magnetic field with $E_z = -Ku_oB_0$, and $0 < K < 1$.[7] Hence, we can combine the electric field into the magnetic field to estimate orders of magnitude for different forces.
6. The channel is not short, which means $\epsilon = 2d/L$ is small.

A proper order estimation for several nondimensional parameters is crucial to simplify the governing equations. Hence, we need first to estimate the orders of magnitude for several nondimensional parameters, including Mach number, $Ma = U_o/\sqrt{\gamma RT_o}$, Reynolds number, $Re = (2d)\rho_oU_o/\mu$, magnetic force number, $R_b = B_0^2/(\rho_oU_o^2\mu_m)$, magnetic Reynolds number, $Re_\sigma = (2d)U_o\mu_o\sigma_o$, Knudsen number, $Kn = \lambda/(2d) \sim \sqrt{\frac{\pi\gamma}{2}}\frac{Ma}{Re}$, and Hartmann number, $H_a = \sqrt{ReR_\sigma R_b}$. These nondimensional numbers are based on the quantities at the outlet, and the external magnetic field.

By choosing the whole flow domain as an integral domain [5] and exercising the X-momentum equation globally, we obtain:

$$2d(P_o - P_i + \rho_oU_o^2 - \rho_iU_i^2) = \mu\frac{U_i + U_o}{2d}2L - \sigma B_0(E_z + \frac{U_o + U_i}{2}B_0)(2d)L$$

Further, we drop those terms with U_i because $U_i << U_o$ for a pressure driven gas flow inside a microchannel. Then, the following simple relation is obtained:

$$\epsilon\big(1 + 1/(\gamma Ma^2)(1 - 1/P)\big) \sim 1/Re + R_\sigma R_b \tag{1}$$

where $P = p_i/p_o$, and $\epsilon = 2d/L$. The parameter K, which is a key physical factor for the microchannel flow, presents the ratio between the electric field and the magnetic field. Here it is considered to be the same order, or less than, the magnetic field effects, and it is combined into the factor $R_\sigma R_b$.

Eqn.(1) contains four nontrivial terms: 1) the momentum change term, represented by ϵ on the left hand side; 2) the pressure drop term applying at the channel inlet and outlet, $\epsilon/(\gamma M_a^2)(1-1/P)$, on the left hand side; 3) the viscous force along the wall surfaces, $1/Re$, on the right hand side; and 4) the magnetic/electric force which applies to the whole domain, $R_\sigma R_b$, on the right hand side. There are many choices to balance these terms: all terms can share the same order of magnitudes; or three terms share the larger order while the other term is smaller; or two of the four terms are larger while the other two terms are equally smaller; or two of the four terms are large, one term is relatively small and the other term is the smallest. Hence, from the above four permutations, there will be at least $1 + C_4^3 + C_4^2 C_2^2 + C_4^2 C_2^1 = 23$ classes of combinations, and most of these combinations can have many detailed subclasses as well. Here we are interested in investigating the interactions among viscous stress, pressure drop and magnetic/electric field effects. Different parameter orders may result in different simplified governing equations and different flow solutions, including hypersonic, supersonic and transonic flows, as we illustrated in our previous paper.[5] Here we are specially interested in slow MGD flows under effects of strong pressure difference, strong viscous effects along channel wall, and strong MGD force. In this study, we select two viable cases with the following parameter combinations which satisfy (1):

1. $Re \sim \epsilon, M_a \sim \epsilon, Kn \sim 1; R_b \sim 1/\epsilon^2, R_\sigma \sim \epsilon;\ H_a \sim 1;$
2. $Re \sim 1, M_a \sim \epsilon^{1/2}, Kn \sim \epsilon^{1/2}; R_b \sim 1/\epsilon, R_\sigma \sim \epsilon;\ H_a \sim 1;$

Previously[5] we showed that the Reynolds and Mach numbers in these two cases render a balance between the viscous effects and the pressure drop term. We intend to set R_σ at least one order smaller than R_b to create a very small induced magnetic field, but the magnetic interaction factor $Q = R_\sigma R_b$ is assumed to be as strong as the terms for viscous force and pressure drop.

3 Asymptotic Solutions

Here the flow quantities are normalized with the averaged properties at the outlet, and the X-, Y-coordinates are normalized with L, $2d$ correspondingly.[1,5] With the two sets of parameters previously selected, the non-dimensional normalized governing equations and boundary conditions are:[10]

$$\epsilon\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0, p = \rho T = nT \tag{2}$$

$$\epsilon\frac{\partial}{\partial x}(\rho uu + \frac{p}{\gamma M_a^2}) + \frac{\partial(\rho vu)}{\partial y} = \frac{1}{Re}(\epsilon^2\frac{4}{3}\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{1}{3}\epsilon\frac{\partial^2 v}{\partial x\partial y}) + Q(K-u) \tag{3}$$

$$\epsilon\frac{\partial(\rho uv)}{\partial x} + \frac{\partial}{\partial y}(\rho v^2 + \frac{p}{\gamma M_a^2}) = \frac{1}{Re}(\epsilon^2\frac{\partial^2 v}{\partial x^2} + \frac{4}{3}\frac{\partial^2 v}{\partial y^2} + \epsilon\frac{\partial^2 u}{\partial x\partial y}) \tag{4}$$

$$\begin{aligned}\epsilon\rho u\tfrac{\partial T}{\partial x} + \rho v\tfrac{\partial T}{\partial y} = \epsilon\tfrac{\gamma-1}{\gamma}u\tfrac{\partial p}{\partial x} + \tfrac{\gamma-1}{\gamma}v\tfrac{\partial p}{\partial y} + \tfrac{1}{RePr}(\epsilon^2\tfrac{\partial^2 T}{\partial x^2} + \tfrac{\partial^2 T}{\partial y^2}) + \tfrac{(\gamma-1)M^2}{Re}\Big[2(\tfrac{\partial u}{\partial x})^2\epsilon^2 \\ +2(\tfrac{\partial v}{\partial y})^2 + (\epsilon\tfrac{\partial v}{\partial x} + \tfrac{\partial u}{\partial y})^2 - \tfrac{2}{3}(\tfrac{\partial u}{\partial x}\epsilon + \tfrac{\partial v}{\partial y})^2\Big] + QK(K-u)\end{aligned} \tag{5}$$

$$u_1(x,y)|_{y=\pm1/2} = \Theta_u Kn(x)|_{y=\pm1/2}\frac{\partial u_1(x,y)}{\partial n}|_{y=\pm1/2} \tag{6}$$

$$T|_{y=\pm1/2} - T_w = \Theta_T\frac{2\gamma}{Pr(\gamma+1)}Kn(x)\big(\frac{\partial T}{\partial n}\big)_{y=\pm1/2} \tag{7}$$

where the relations $|v| < |u|$ and $|B_x| << |B_0|$ are used to simplify the source terms in the momentum and energy equations, $\Theta_u = (2 - \sigma_u)/\sigma_u$ and $\Theta_T = (2 - \sigma_T)/\sigma_T$. In (6) for the nondimensional velocity boundary condition, a temperature gradient term is omitted because it is smaller than the velocity gradient with the two specific groups of nondimensional parameters.

Further, we assume the following expansions $u = u_1 + \epsilon u_2 + ..., v = v_1 + \epsilon v_2 + ..., p = p_1 + \epsilon p_2 + ..., \rho = \rho_1 + \epsilon \rho_2 + ..., n = n_1 + \epsilon n_2 + ..., T = 1 + \epsilon T_2$. The last one implies a quasi-isothermal assumption.[5] Specifically, from $p(x, y) = \rho(x, y)(1 + \epsilon T_2(x, y))$, we obtain $p_1 = \rho_1$ as the zero order relation and the isothermal assumption proposed by Arkilic *et al*[1] is relaxed.

For the two sets of parameters, the Y-momentum equation simplifies as:

$$\frac{\partial p_1(x,y)}{\partial y} = 0 \tag{8}$$

With the low magnetic Reynolds number assumption, the magnetic term is at least one order smaller than the pressure term. From (8) we have $p_1(x, y) = p_1(x)$, which greatly simplifies the following derivations.

The simplified X-momentum equation is:[6]

$$\frac{\partial^2 u_1}{\partial y^2} - H_a^2 u_1 = -H_a^2 K + \frac{\epsilon Re}{\gamma M_a^2}\frac{dp_1}{dx} \tag{9}$$

The slip wall boundary conditions result in different solutions from the classical Hartmann flow solutions. The final U-velocity solution is,

$$u_1(x,y) = C_1 \frac{\epsilon Re}{\gamma M^2}\frac{dp_1}{dx}\cosh(H_a y) - \frac{\epsilon Re}{H_a^2 \gamma M^2}\frac{dp_1}{dx} + K \tag{10}$$

where $C_1 = \frac{1}{H_a^2 C_2} - \frac{K\gamma M^2}{C_2 \epsilon Re(dp_1/dx)}$, $C_2 = \cosh(\frac{H_a}{2}) + \Theta_u Kn(x)\sinh(\frac{H_a}{2})H_a$.

The zero-order solution for the V-velocity is obtained from the zero-order of the continuity equation, $v_1 = 0$, and the next order can be obtained from the continuity equation, by using the relation $\rho_1 = p_1$:

$$\frac{\partial(p_1(x)u_1(x,y))}{\partial x} + \frac{\partial(p_1(x)v_2(x,y))}{\partial y} = 0$$

and the result is:

$$\begin{aligned} v_2(x,y) = {} & A_1 \sinh(H_a y)[-\tfrac{1}{2}\tfrac{d^2 p_1^2}{p_1 dx^2} + A_2 \tfrac{d^2 p_1}{p_1 dx^2}] + A_3 K \tfrac{dp_1}{p_1 dx}\sinh(H_a y) \\ & + y \tfrac{\epsilon Re}{2H_a^2 \gamma M^2}\tfrac{d^2 p^2}{p_1 dx^2} - \tfrac{dp_1}{p_1 dx}Ky \end{aligned} \tag{11}$$

where $A_1 = \frac{\epsilon Re}{\gamma M^2 H_a^3 \cosh(H_a/2)}$, $A_2 = \Theta_u H_a Kn_o \tanh(H_a/2)$, $A_3 = \frac{1}{H_a \cosh(H_a/2)}$. The temperature distribution can be obtained by the following simplified energy equation and the temperature-jump wall boundary conditions:

$$\frac{1}{RePr}\frac{\partial^2 T_2}{\partial y^2} = -\frac{\gamma - 1}{\gamma}u_1\frac{dp_1}{dx} - \frac{(\gamma-1)M^2}{\epsilon Re}\left(\frac{\partial u_1(x,y)}{\partial y}\right)^2 - QK(K-U) \tag{12}$$

The solution for the temperature field is:

$$T_2(x,y) = RePr\left(\frac{N_3}{H_a^2}\cosh(H_a y) + \frac{N_5}{4H_a^2}\cosh(2H_a y) + \frac{N_6}{2}y^2 + N_8\right) \tag{13}$$

where N_8 is determined by the temperature jump boundary condition:

$$N_8 = \tfrac{T_w-1}{\epsilon RePr} - \tfrac{N_3}{H_a^2}\cosh\left(\tfrac{H_a}{2}\right) - \tfrac{N_5}{4H_a^2}\cosh(H_a) - \tfrac{N_6}{8} - \Theta_T \tfrac{2\gamma Kn(x)}{Pr(\gamma+1)} \\ [\tfrac{N_3}{H_a}\sinh\left(\tfrac{H_a}{2}\right) + \tfrac{N_5}{2H_a}\sinh(H_a) + \tfrac{N_6}{2}]$$

Moreover $N_1 = C_1 \frac{\epsilon Re}{\gamma M^2}\frac{dp_1}{dx}$, $N_2 = -\frac{\epsilon Re}{H_a^2\gamma M^2}\frac{dp_1}{dx} + K$, $N_3 = -\frac{\gamma-1}{\gamma}N_1\frac{dp_1}{dx} + QKN_1$, $N_4 = -\frac{\gamma-1}{\gamma}N_2\frac{dp_1}{dx} - QK^2 + QKN_2$, $N_5 = -\frac{(\gamma-1)M^2}{2\epsilon Re}N_1^2H_a^2$, $N_6 = N_4 - N_5$. The density distribution is approximated as $\rho(x,y) = p_1(x)/(1+\epsilon T_2(x,y)$. It is evident that the density is not constant at any specific station with x=constant.

The pressure distribution is crucial to the whole set of solution since $p(x)$ and its gradients dominate the coefficients for $u(x,y)$, $v(x,y)$ and $T(x,y)$. Evaluating the V-velocity along the upper channel wall results in the following equation:

$$D_1\frac{d^2p_1^2}{dx^2} + D_2\frac{d^2p_1}{dx^2} + D_3K\frac{dp_1}{dx} = 0 \tag{14}$$

where $D_1 = -\left(\frac{\tanh(H_a/2)}{2H_a^3} - \frac{1}{4H_a^2}\right)\frac{\epsilon Re}{\gamma M^2}$, $D_2 = \frac{\tanh^2(H_a/2)}{H_a^2}\frac{\epsilon Re}{\gamma M^2}\Theta_u Kn_o$ and $D_3 = -\frac{1}{2} + \frac{\tanh(H_a/2)}{H_a}$. The boundary conditions, $p_1(0) = P$ and $p_1(1) = 1$ lead to the following exact solutions to the nonlinear ordinary differential equation:

1. if $K = 0$, i.e., in the absence of the electric field effect:

$$p_1(x) = \frac{-D_2 + \sqrt{D_2^2 + 4D_1(D_1P^2 + D_2P + D_4x)}}{2D_1} \tag{15}$$

   where $D_4 = D_1(1-P^2) + D_2(1-P)$.
2. if $K > 0$, i.e., with a uniform electric field:

$$2D_1p_1(x) + (\frac{2D_1G_1}{D_3K} + D_2)\ln[D_3Kp_1(x) - G_1] = -xD_3K + G_2D_3K \tag{16}$$

   The two boundary conditions exactly determine the coefficients G_1 and G_2 in the above solution: $G_2 = \frac{2D_1}{D_3K} + (\frac{2D_1G_1}{D_3^2K^2} + \frac{D_2}{D_3K})\ln(D_3K - G_1) + 1$ and $2D_1(P-1) - D_3K = (\frac{2D_1G_1}{D_3K} + D_2)\ln\left(\frac{D_3K-G_1}{D_3KP-G_1}\right)$. Solving for G_1 requires an iterative method. However, for a given set of non-dimensional numbers, G_1 and G_2 are completely determined.

4 Numerical Validations

We perform numerical computations to solve the low magnetic Reynolds number MGD equations, and compare the results with the corresponding analytical results. The computations are performed with a well-tested general Navier-Stokes equation solver. The flow parameters are: $\epsilon = 0.06$, $L = 20\ \mu m$, $p_1 = 2\ atm$, $p_2 = 1\ atm$, $T_1 = 300\ K$, $\sigma_u = \sigma_T = 0.85$, oxygen gas, $\gamma = 1.4$, $Pr = 0.72$, $\mu = 1.919 \times 10^{-5}\ sec.N/m^2$, wall temperature $T_w = 300\ K$, $R_b = 1/\epsilon$ and $R_\sigma = \epsilon$. When the magnetic field is enabled, K is chosen to be 0.0, 0.5 or 0.9. Many of these simulation parameters are the same as those in our previous paper[5] on a neutral gas flow case.

Figures 2-3 correspond to a case with $K = 0.5$, the so-called "impedance match" case for Hartmann flow.[7] Figures 2 shows comparisons of U-velocity results, where the velocity-slip effects are evident along the wall while some discrepancies are shown at the outlet boundary. Figure 3 shows comparisons of numerical and analytical temperature results. In general, the temperature fields have the same trends with large discrepancies. Figures 4-6 are based on four different cases: 1) no magnetic and electric fields; 2) $K = 0$; 3) $K = 0.5$; and 4) $K = 0.9$. We intend to show some trends by comparing the results from these four cases. Nonuniform strength of sources terms are added to the X-momentum and the energy equations for the last two cases. Figure 4 shows the pressure distributions along the flow direction, with the linear pressure distributions subtracted out. The lines without soiled symbols are analytical results, while those with solid symbols represent the numerical simulation results for the last case. Though numerical solutions are obtained for all cases, here we merely present one case of $K = 0.9$ for clarity. It is clear that the linear pressure gradient assumed in the Hartmann flow is not applicable here and it may result in inaccuracies if used carelessly. For these four test cases, at any specific station pressure level increases from case 1 to case 4, indicating that the extra magnetic field or electric field results in increasingly stronger impedance to the flow field. For Cases 1 and 2, the numerical and analytical results are essentially the same; while for the last two cases, the nonlinearities become more appreciable. Larger discrepancies are found for cases 3 and 4 than those of cases 1 and 2. Figure 5 shows the different U-velocity profiles for the four test cases at the middle station of the channel, $x/L = 0.5$. The electric and magnetic fields are shown to strongly influence the velocity profiles. Numerical and analytical results are close. For clarity, only the numerical results of case 4 are shown. Figure 6 displays the profiles of velocity

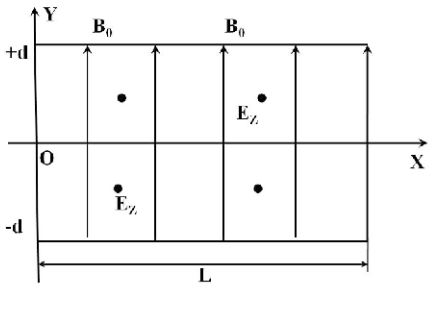


Fig. 1. Illustration of the problem, $B = (0, B_0, 0)$, $E = (0, 0, E_z)$

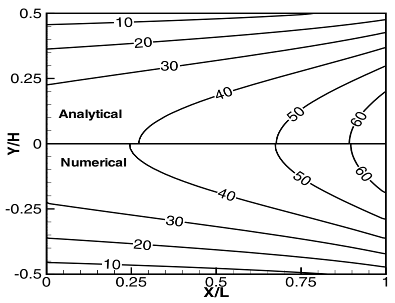
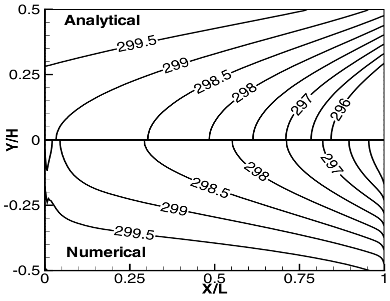


Fig. 2. Contours of U-velocity, $P = 2$, $K = 0.5$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$



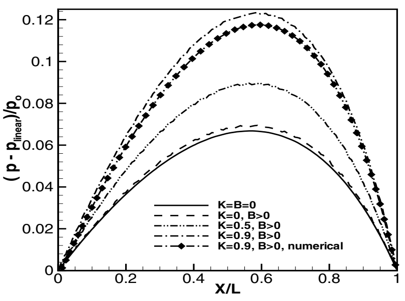


Fig. 3. Contours of temperature, $P = 2$, $K = 0.5$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$

Fig. 4. Pressure distribution along the flow direction, $P = 2$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$

slips along the upper walls for these four test cases. In general, the slip velocities increase as gas flows downstream, and numerical results are found to agree with analytical results. Essentially, for the last two cases, the gas flows are both Fanno and Rayleigh flows due to the wall friction and Joule heating effects. Hence, with stronger Joule heating effects in case 4, the subsonic flow increases its speed more significantly towards the sonic speed than the that of case 3. Now, we will examine the discrepancies found on the results. 1) The numerical simulation results are from the whole MGD equations, while the analytical solutions are based on the simplified asymptotic equations. If we consider the truncation error $O(\epsilon)$ as an upper limit, there is about 6% theoretical difference with $\epsilon = 0.06$; 2) The analytical results are actually based on the assumption that the channel is in the middle of flow field without entrance effects; however, for numerical solutions, the entrance effects are not avoidable due to the presence of significant friction occurring at the inlet region. The uniform free stream needs to "adjust" to the channel flow around the inlet entrance. The inlet and outlet boundary condition treatment is very subtle, and the outlet boundary condition treatment is important since the gradients there are large. However, these end boundary effects are not included in the asymptotic solutions at all. Significant discrepancies therefore show up in the V-Velocity and temperature profiles. If we use further longer simulation case, most probably the results shall be better. 3) Betweens the analytical and numerical solutions, the pressure fields already have some discrepancies, and with the small length dimension, dp/dx and dp^2/dx^2 must be huge quantities. For the temperature field, the dp/dx term dominates in the coefficients, and $(dp/dx)^2$ appears in the coefficients N_5 and N_6. Hence, it is not surprising to see that the temperature profiles have poorer agreement than the velocity and pressure results. 4) Analytically, here we only consider the leading term for the electric and magnetic field effects but since they are coupled in the source terms for the momentum and energy equations, some nonlinear effects are possible. Figures 7 and 8 show the temperature profiles and the temperature gradients along the upper wall boundary. As discussed earlier, it is very

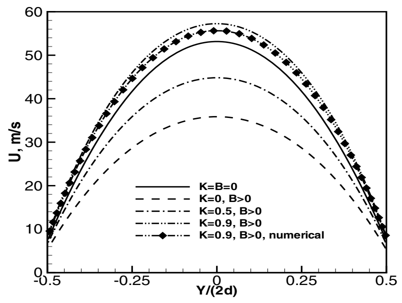


Fig. 5. Profiles of U-velocity distributions at the middle section, $x/L = 0.5$, $P = 2$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$

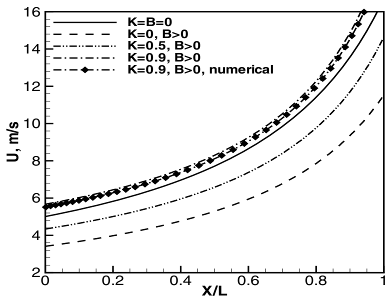


Fig. 6. Profiles of slip velocity along the wall, $y = d$, $P = 2$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$

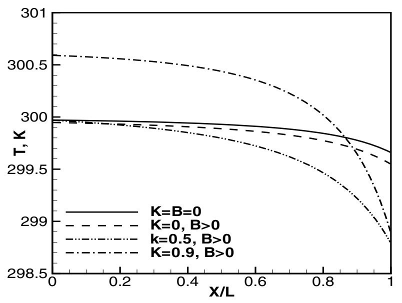


Fig. 7. Profiles of temperature distributions along the upper wall, $P = 2$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$

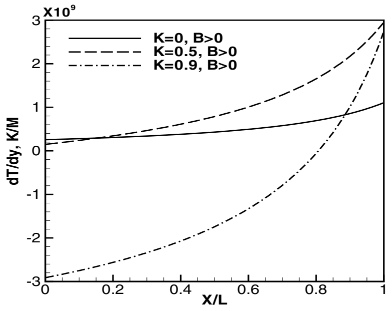


Fig. 8. Profiles of temperature gradient along the upper wall, $P = 2$, $R_b = 1/\epsilon$, $R_\sigma = \epsilon$, $\epsilon = 0.06$

difficult to obtain accurate temperature profiles, especially when both electric and magnetic fields are considered. Hence, these plots only show the analytical results. For cases $K = 0.5$ and $K = 0.9$, there are Joule heating effects in the energy equation. For $K = 0.9$, stronger Joule heating deposits more energy into the field, resulting in a much higher increase in the temperature field. As shown in Fig. 7, a portion of gas close to the wall is actually hotter than the wall. Correspondingly, for the last case, heat flux is transferred into the wall instead from the wall, as shown by Fig. 8. There, it also indicates one important effect associated with an MGD gas flow inside a microchannel: along the wall, even when the temperature change is very small, the normal temperature gradient can be huge because of the narrow channel height.[5] Hence, this shows that the heat transfer problem in microchannels is of practical importance. For Case 1 (K=0) without any electric and magnetic field effects, the temperature gradient reaches an order of 1×10^6 K/m. By contrast, with magnetic and electric field effects, (K=0.9), the magnitude of temperature gradient becomes much larger and variant, as shown in Fig.8.

5 Conclusion

We reported an analysis of rarefied MGD gas flows inside a 2D microchannel with velocity-slip and temperature-jump boundary conditions assuming that the magnetic Reynolds number is low and the flow is quasi-isothermal. By carefully comparing different orders of magnitude for the pressure drop, viscous shear stress at the channel wall, and the magnetic forces, two sets of parameters are selected and used to simplify the MGD equations. This study yields asymptotic solutions for velocity components, pressure and temperature. With stronger Joule heating effects, the Rayleigh process effects become significant, and the average pressure, velocity and temperature inside the channel increases. In general, the pressure gradient along the flow direction is nonlinear inside the channel, and the velocity and density distributions are nonuniform. Numerical solutions of the same formulation are obtained to validate these asymptotic solutions; explanations are provided for the discrepancies found between these solutions.

Future work may include other asymptotic solutions with different set of parameters, and the corresponding solutions to three-dimensional or axially symmetric flows.

References

1. Arkilic, E.B., Schmidt, M.A., Breuer, K.S.: Gaseous Slip Flow in Long Microchannels. J. Microelectromech. Syst. 6(2) (1997)
2. Zohar, Y., Lee, S.Y.K., Lee, W.Y., Jiang, L., Tong, P.: Subsonic Gas Flow in a Straight and Uniform Microchannel. J. Fluid Mech. 472 (2002)
3. Cai, C., Boyd, I., Fan, J., Candler, G.V.: Direct Simulation Methods for Low-Speed Microchannel Flows. J. Thermophys Heat Trans. 14 (2000)
4. Xu, K., Li, Z.: Microchannel Flow in the Slip Regime: Gas-Kinetic BGK-Burnett Solutions. J. Fluid Mech. 513 (2004)
5. Cai, C., Sun, Q., Boyd, I.D.: Gas Flows in Microchannels and Microtubes. J. Fluid Mech. 589 (2007)
6. Boyd, T.J.M., Sanderson, J.J.: Plasmadynamics, Nelson, London (1969)
7. Gaitonde, D.V., Poggie, J.: Simulation of MHD Flow Control Techniques. AIAA paper 2000-2326
8. Agarwal, R.K.: Lattice Boltzmann Simulations of Magnetohydrodynamics Slip Flow in Microchannels. AIAA paper 2005-4782
9. Naterer, G.F., Camberos, J.A.: Entropy Based Design of Thermofluid and Microfluidc Systems. CRL Pr I Llc (June 2007)
10. Cai, C., Boyd, I.D.: Compressible Gas Flows in a Two-Dimensional Planar Microchannel. J. Thermophys Heat Transs. 21(3), 608–615 (2007)

Practical Numerical Simulations of Two-Phase Flow and Heat Transfer Phenomena in a Thermosyphon for Design and Development

Zheshu Ma[1], Ali Turan[2], and Shengmin Guo[2]

[1] Department of Power Engineering, Jiangsu University of Science and Technology, Zhenjiang, 212003, China
[2] School of Mechanical, Aerospace and Civil Engineering, The University of Manchester, Manchester, M60 1QD, UK
mazheshu@126.com, A.Turan@manchester.ac.uk, sguo2@lsu.edu

Abstract. Two-phase closed thermosyphons are widely-used in various energy intensive industries including chemical, transportation and other industries for their inherently high heat transfer efficiency, simple construction and reliable operational performance. However, the performance parameters of two-phase closed thermosyphons including the distribution of internal pressure, steam and liquid phase mass fractions, velocity and wall temperatures are obtained primarily via experimental investigations. In this paper numerical methods are discussed and a two-fluid model is employed to describe the two-phase flow and heat transfer processes in a two-phase closed thermosyphon. The IPSA (Inter Phase Slip Algorithm) algorithm is employed to solve the coupled interactions of steam and liquid phases along the phase interface. Flow patterns and distribution of parameters under different conditions are predicted with numerical results agreeing well for the most part with experimental results. Thus, the numerical method and solution procedures are claimed to be of practical utility and can in essence be used profitably to simulate flow and heat transfer phenomena in thermosyphons and other types of heat pipes.

Keywords: IPSA algorithm, two-phase flow, numerical simulation, thermosyphon.

1 Introduction

With the integration of electronic circuits and greatly increasing power capacity of power appliances, cooling methods with improved efficiency that can provide suitable thermal environments for electronic equipments are required, as electronic components are constantly being miniaturized in size. Today, electronic cooling technologies have become one of the key considerations in design and operation of electronic equipment [1-3]. State-of-the-art cooling technologies and developments related to high power electronic devices include a variety of processes ranging from phase-change cooling, forced convection cooling, natural convection cooling and micro heat exchanger cooling [4]. Heat pipe electronic cooling is a kind of phase-change cooling technology possessing several striking advantages, i.e., a large equivalent thermal conductance, excellent packaging flexibility, passive operation and high reliability [4].

Heat pipes including two-phase closed thermosyphons are a promising phase-change electronic cooling technology for the various advantages offered including high heat transfer efficiency, simple construction and reliable operational performance. In order to analyze their running performance and to further design more efficient and reliable two-phase closed thermosyphons, designers and researchers must possess a sound understanding of the internal two phase flow details and heat transfer characteristics in a thermosyphon, in particular looking at the subcooled boiling process for the liquid including vapor release from the saturated liquid, vapor condensation back to the liquid, upward flow of vapor and downward flow of condensed liquid, and different flow regimes varying according to internal flow characteristics as regulated by the operational conditions of thermosyphons. Since the inherent internal flow and heat transfer details are very complicated, the performance parameters of two-phase closed thermosyphons including the distribution of internal pressure, steam and liquid phase mass fractions, velocity and wall temperatures are currently primarily obtained and even designed by exclusively devising appropriate experimental studies.

Computational simulation procedures employing advanced engineering mathematical and physical models via high performance computers have been playing more and more important roles in most engineering fields for the inherent advantages offered including efficiency and accuracy, especially for cases where measurements are impractical. Specifically, regarding the numerical simulation of two phase flow phenomena within heat pipes, state-of-the-art solutions initially analyze the internal vapor-liquid two phase flow regimes, to subsequently identify flow characteristics for the different flow zones and finally appropriate mathematical and physical models are numerically solved.

In this paper, a specific numerical method including a two-fluid model is employed to describe the two-phase flow and heat transfer processes in a two-phase closed thermosyphon. Widely used IPSA algorithm [5-7, 9] is employed to solve the coupled interaction of steam and liquid phases along the phase interface. Flow patterns and distribution of parameters under different conditions are predicted. Numerical results agree well with experimental results.

2 Mathematical Model and Numerical Method

To numerically simulate the complex heat transfer and two-phase flow phenomena in the thermosyphon configuration studied , a two-fluid model is employed. The two-fluid model provides an effective means of representing the coexistence of laminar and turbulent flows. It considers the system to be composed of two fluids that coexist simultaneously in time and space but possess different volume fractions and can naturally reflect the exchange of mass, momentum and energy between the two fluids and is well suited to describe the whole computational domain.

When the two-fluid model is applied to predict the thermo-fluid details within the thermosyphon, the following assumptions are naturally made to achieve a workable numerical model: (1)The heat conduction resistance between the outer surface of the pipe and the inner wall of the pipe is neglected; (2)The thermodynamic properties of the working vapor and liquid are assumed to be constant ,i.e., vapor and liquid phases are saturated; (3)Vapor and liquid flows are laminar and incompressible; (4)The evaporation and condensation processes only occur at the vapor-liquid interface.

The basic governing equations in two dimensions describing the heat transfer and flow of the vapor phase within the studied thermosyphon are provided in Eqs (1) to (4).

Continuity equation for the vapor:

$$\frac{\partial}{\partial x}(R_1\rho_1 u_1)+\frac{\partial}{\partial y}(R_1\rho_1 v_1)=\dot{m}_{21} \tag{1}$$

Momentum equations for the vapor:

$$\frac{\partial}{\partial x}\left(R_1\rho_1 u_1^2\right)+\frac{\partial}{\partial y}(R_1\rho_1 u_1 v_1)=-R_1\frac{\partial p}{\partial x}+R_1\mu_1\frac{\partial^2 u_1}{\partial y^2}+\dot{m}_{21}|u_1-u_2|-F_{21}-R_1\rho_1 g \tag{2}$$

$$\frac{\partial}{\partial x}(R_1\rho_1 u_1 v_1)+\frac{\partial}{\partial y}\left(R_1\rho_1 v_1^2\right)=-R_1\frac{\partial p}{\partial y}+R_1\mu_1\frac{\partial^2 v_1}{\partial x^2}+\dot{m}_{21}|v_1-v_2| \tag{3}$$

Energy equation for the vapor:

$$\frac{\partial}{\partial x}(R_1\rho_1 u_1 h_1)+\frac{\partial}{\partial y}(R_1\rho_1 v_1 h_1)=R_1\frac{\mu_1}{\mathrm{Pr}_1}\left(\frac{\partial^2 h_1}{\partial y^2}\right)+\dot{m}_{21}|h_1-h_2|+J_c \tag{4}$$

The basic governing equations in two dimensions describing the heat transfer and the flow of the liquid phase within the thermosyphon are given in Equations (5) to (8).

Continuity equation for the liquid:

$$\frac{\partial}{\partial x}(R_2\rho_2 u_2)+\frac{\partial}{\partial y}(R_2\rho_2 v_2)=-\dot{m}_{21} \tag{5}$$

Momentum equations for the liquid:

$$\frac{\partial}{\partial x}\left(R_2\rho_2 u_2^2\right)+\frac{\partial}{\partial y}(R_2\rho_2 u_2 v_2)=-R_2\frac{\partial p}{\partial x}+R_2\mu_2\frac{\partial^2 u_2}{\partial y^2}-\dot{m}_{21}|u_1-u_2|+F_{21}-R_2\rho_2 g \tag{6}$$

$$\frac{\partial}{\partial x}(R_2\rho_2 u_2 v_2)+\frac{\partial}{\partial y}\left(R_2\rho_2 v_2^2\right)=-R_2\frac{\partial p}{\partial y}+R_2\mu_2\frac{\partial^2 v_2}{\partial x^2}-\dot{m}_{21}|v_1-v_2| \tag{7}$$

Energy equation for the liquid:

$$\frac{\partial}{\partial x}(R_2\rho_2 u_2 h_2)+\frac{\partial}{\partial y}(R_2\rho_2 v_2 h_2)=R_2\frac{\mu_2}{\mathrm{Pr}_2}\left(\frac{\partial^2 h_2}{\partial y^2}\right)-\dot{m}_{21}|h_1-h_2|-J_c+\frac{C_2^w}{A_2}\dot{Q} \tag{8}$$

In equations (1) to (8), x and y are axial coordinate and diametrical coordinate, respectively; R_1 and R_2 are the volume fractions of vapor phase and liquid phase respectively; $\dot{m}_{21}$ is the interphase mass transfer rate between the vapor and liquid , (kg/(m^3.s)). u_1 and v_1 are axial and radial velocity of the vapor phase respectively, (m/s) while u_2 and v_2 denote the axial and radial velocity of the liquid, (m/s). μ_1 and μ_2 are the dynamic viscosity coefficients for the vapor and liquid (N.s/m^2), while h_1 and h_2 signify the specific enthalpies of the vapor and liquid, respectively (kJ/kg). ρ_1 and ρ_2 are the densities of the vapor and liquid, (kg/m^3). F_{21} refers to the interphase friction between the vapor and liquid on the phase interface, (N). J_c is heat transfer rate between the vapor and liquid (W/s) while $\dot{Q}$ refers to the heat addition to the thermosyphon, (W/s) with C_2^w denoting the length of the wetted perimeter, (m). Finally, A_2 is the contact area of the liquid phase with the pipe wall, (m^2) and $\Pr_1$ and $\Pr_2$ are the Prandtl numbers for the vapor and liquid phases respectively.

During the numerical calculation, R_1 and R_2 must satisfy Eq. (9).

$$R_1 + R_2 = 1.0 \tag{9}$$

The mass transfer rate between the vapor and liquid phases, $\dot{m}_{21}$, can be described as given by Eq. (10).

$$\dot{m}_{21} = \frac{J_c}{h_{fg}} \tag{10}$$

where h_{fg} is the vaporization latent heat of the working liquid, in kJ/kg.

Heat transfer rate between vapor phase and liquid phase, J_c, is defined as given by the sum of vapor phase heat transfer rate J_{c1} and the liquid phase heat transfer rate J_{c2}.

$$J_c = J_{c1} + J_{c2} \tag{11}$$

$$J_{c1} = c_2 R_1 R_2 \rho_2 \left(h_1 - h_{1,s}\right) \tag{12}$$

$$J_{c2} = c_3 R_1 R_2 \rho_2 \left(h_2 - h_{2,s}\right) \tag{13}$$

Friction force between the vapor phase and the liquid phase, F_{21}, can be written as in Eq. (14).

$$F_{21} = c_1 R_1 R_2 \rho_2 \left(u_2 - u_1\right) \tag{14}$$

Here c_1, c_2 and c_3 in Eq.(11)-Eq.(14) are different empirical constants, while $h_{1,s}$ and $h_{2,s}$ are the specific enthalpy of the vapor phase and the liquid phase along the phase interface respectively.

A conservative finite-volume method with a staggered variable arrangement [7] is used to solve the nonlinear, coupled system of partial differential equations outlined in the previous section. The solution domain is subdivided into a large number of differential control volumes for the scalar variables and velocities. Nodal values of the scalar quantities are located at the geometric centres of their control volumes while the velocities are stored on the scalar control volume faces. Temporal derivatives are discretized by a fully implicit first order backward Euler scheme. Regarding the present cases investigated, the equations arising from the discretization procedure can be assembled in the form of a quasi-tridiagonal matrix which is solved via an Alternating Direction Implicit (ADI) version of the direct Thomas-algorithm. The discretized equations are numerically coupled and the solution is henceforth obtained by employing the widely available iterative and segregated IPSA-solver methodology developed by Spalding [7]. At each time step, the volume fractions and the velocity fields are initially predicted using the previous time step pressure field, which are then corrected via the solution of a "pressure correction equation" derived from the overall mass conservation requirement. The values of pressure and velocities are subsequently updated to satisfy mass-conservation to yield converged values to be used as the initial field for recalculating the flow variables at the next level of iteration. The IPSA-procedure is iteratively repeated until a pre-set convergence criterion is met. As an additional convergence enhancement, the Partial Elimination Algorithm (PEA) is also employed. Convergence of the method is generally declared as the sum of the absolute residual norms falls below 0.001 for all the variables.

3 Results and Discussions

The thermosyphon studied in this paper employed water as the working media and the heat pipe was made of bronze [10]. The thermosyphon was 1.1 meters long and its inner diameter and wall thickness are 25mm and 2.5mm respectively. According to the relevant operational characteristics as validated upon performing appropriate measurements, the lengths of evaporator and condenser sections, including the isothermal region are given as: the axial length of the evaporator section is about 510mm, while the axial length of condenser section is specified to be 420mm. Finally, the axial length of isothermal section is assumed to be 170mm. Heat addition density is specified as 10~60kW/m^2.

Fig.1 indicates the average wall temperature in the evaporator section under different heat addition densities. With the increase of heat addition density which signifies a higher total heat transfer quantity, the average wall temperature in evaporator section increases. From Fig.1, one can see the computational results agree well with the experimental results. The computational field also suggests a linear relationship regarding the variation of the average temperature in evaporator section versus the heat addition density. The experimental results which deviate somewhat from the predicted computational line are suggestive of external influences brought on by the particular experimental procedures, the test setup and other operational and measurement inaccuracies. Such factors arise mainly from practical heat transfer considerations that could not be strictly prevented using the current experimental setup to provide for rigorous adiabaticity in the adiabatic section of the working hardware.

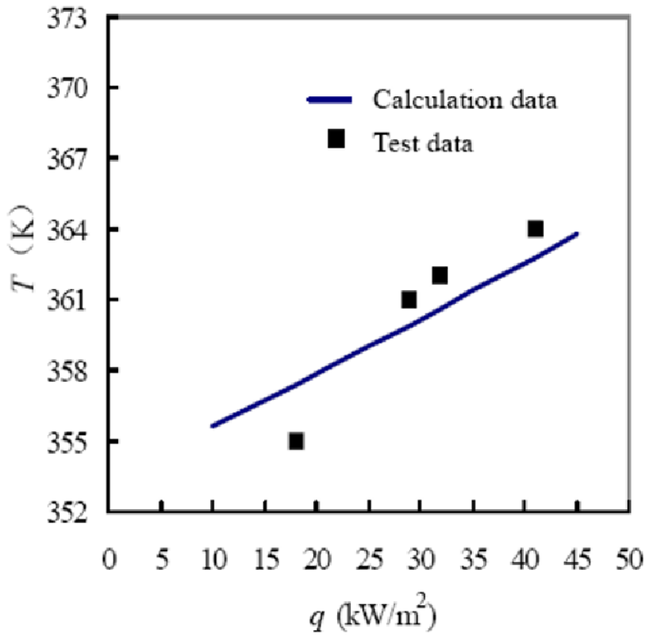


Fig. 1. Wall temperature in the evaporator section varying with heat addition density

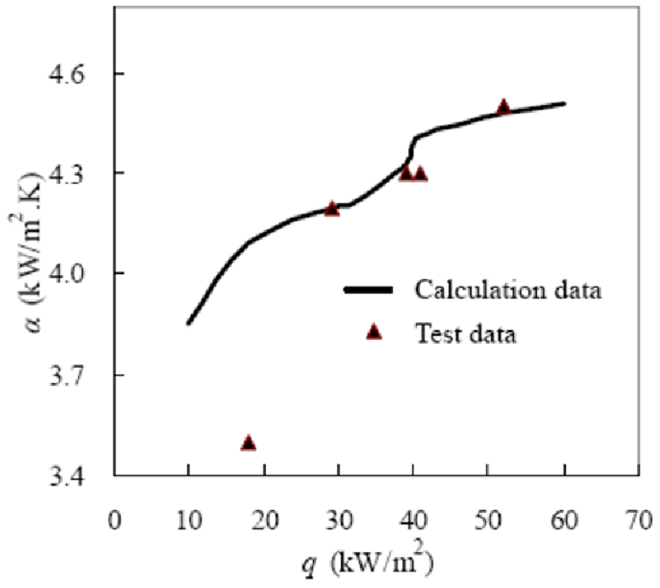


Fig. 2. Heat transfer coefficient varying with heat addition density

Fig.2 indicates the heat transfer coefficient in evaporator section under different heat addition densities. With the increase of heat addition density signifying a higher total heat transfer quantity, the heat transfer coefficient in the evaporator section increases. As different from Figure 1, the relationship between the heat transfer coefficient and the heat addition density is non-linear. The calculation results fit the experimental results better under the higher heat addition densities ($40kW/m^2 < q < 60kW/m^2$) as opposed to the lower heat values ($10kW/m^2 < q < 38kW/m^2$). Within the experimental range of heat addition densities, the heat transfer coefficient for the evaporator section of the thermosyphon could reach as high as $4.6kW/m^2.K$.

Using the particular numerical methodologies adopted here, one can gain additional relevant information for the heat transfer and two-phase flow details in the thermosyphon which generally are not accessible by experimental means. Fig.3 depicts the axial velocity distribution of the vapor phase under different heat addition densities.

With the increase of heat addition density, the axial velocity of the vapor increases. For a certain heat addition density value, the axial velocity increases gradually from zero in the evaporator section while it decreases gradually to zero in the condenser section and reaches a maximum value in the adiabatic section. The axial velocity of vapor could reach as high as 10m/s when the heat addition density is given as 1.97kW, a result predicted to cause substantial flow induced vibration and turbulence.

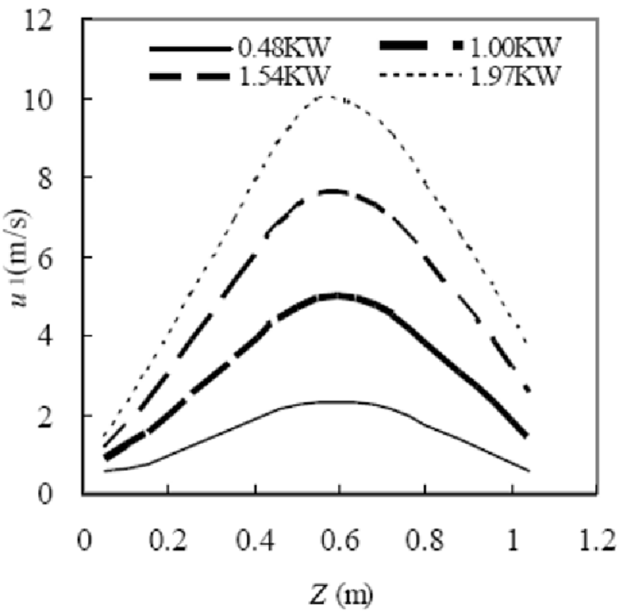


Fig. 3. Axial velocity distribution of vapor phase varying with heat addition density

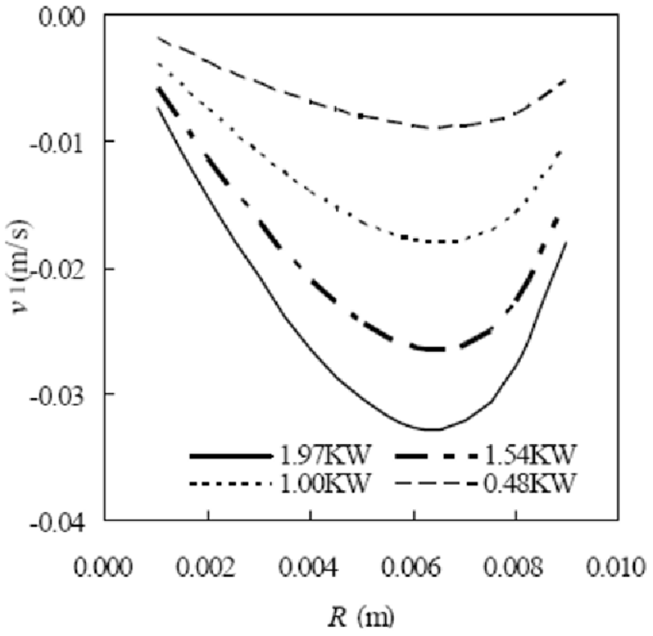


Fig. 4. Radial velocity distribution of vapor phase in evaporator section varying with heat addition density

Fig.4 indicates the radial velocity distribution for the vapor phase in the evaporator section under different heat addition densities. In this particular set of circumstances, the radial velocity of the vapor from the center initially increases from zero and there exists a maximum radial velocity. The radial velocity values for the vapor increase with the increase of the heat addition density. From Figures 3 and 4, one can draw the conclusion that axial velocity of vapor is considerably larger than the radial component ; however, since the radial velocity displays a larger variation vis a vis the axial, here it is shown conclusively that the flow field details in a typical thermosyphon are indeed very complex and numerical simulations henceforth provides for more detailed design and development information.

4 Conclusions

In this paper models to numerically simulate the heat transfer and two-phase flow details within a certain thermosyphon design are presented. Based on the widely employed SIMPLE and IPSA algorithms, successful computations highlighting the inherent operational characteristics of a typical design are carried out. Specifically, relevant flow and heat transfer details are revealed as influenced by the external operational working conditions. Furthermore, the celebrated IPSA algorithm as employed in the current effort strictly indicates that it can profitably and effectively be used to solve two phase flow problems including phase change.

The predicted results agree well on the whole regarding the influence of operational parameters on the performance of the particular thermosyphon geometry. The linear relationship between the average temperature in the evaporator section and the heat addition density is conclusively borne out while the relationship involving the heat transfer coefficient and the heat addition density turns out to be highly non-linear. Within the experimental range of heat addition densities, the heat transfer coefficient for the evaporator section of the thermosyphon could reach as high as 4.6kW/m2.K. The axial velocity of the vapor increases with the increase of heat addition density. The axial velocity of the vapor is substantially larger than the radial component; however, since the radial velocity variation is shown to be substantial vis a vis the axial, here it is shown conclusively the flow details in a typical thermosyphon are indeed extremely complex and hence appropriate numerical simulations provide vital design and development information seriously lacking for innovative hardware design exercises.

Since only a limited number of numerical simulation studies on the internal vapor-liquid two-phase flow and heat transfer details in heat pipes and thermosyphons have been published [11-15], the numerical methodologies employed and successfully used in this paper can be used as a basis for further studies and hopefully in designing reliable hardware development correlations.

Acknowledgments. This work is financially supported by the Qing-Lan Project of Jiangsu Province for universities' outstanding youth skeleton teachers under contract No.161220605 and the Startup-grant of Jiangsu University of Science and Technology for academic research under contract No.2005JD009J. The corresponding author also owes great thanks to School of Mechanical, Aerospace and Civil Engineering in the University of Manchester for providing excellent collaborative research facilities.

References

1. Sauciuc, B.: Design and Testing of the Super Fiber Heat Pipes for Electronics Cooling Applications. In: Annual IEEE Semiconductor Thermal Measurement and Management Symposium, pp. 27–32. IEEE Press, New York (2001)
2. Sathe, A.: Review of Recent Developments in Some Practical Aspects of Air-cooled Electronic Packages. Journal of Heat Transfer, Transactions ASME 120, 830–838 (1998)
3. Ryan, J.M., Roshan, J., Song, L.: Integrated Thermal Management Techniques for High Power Electronic Devices. Applied Thermal Engineering 24, 1143–1156 (2004)
4. Zuo, Z.J., North, M.T., Wert, K.L.: High Heat Flux Heat Pipe Mechanism for Cooling of Electronics. IEEE Transactions on Components and Packaging Technologies 24, 220–225 (2001)
5. Lahey, R.: The Analysis of Phase Separation and Phase Distribution Phenomena Using Two-fluid Models. Nucl.Engng. Des. 122, 17–25 (1990)
6. Ishii., M.: Thermo-fluid Dynamic Theory of Two-phase Flow. Eyrolles Book Publishing House (1975)
7. Spalding, D.B., Markatos, N.C.: Computer Simulation of Multi-Phase Flows. CFDU. Imperial College UK (1983)
8. Shiraishi, M., Kikuchi, K.: Investigation of Heat Transfer Characteristics of Two-phase Closed Thermosyphon. In: Advances in Heat Pipe Technology, pp. 95–104 (1981)

9. Kaya, T., Goldak, J.: Three Dimensional Numerical Analysis of Heat and Mass Transfer in Heat Pipes. Heat And Mass Transfer 43, 775–785 (2007)
10. Xia, J.L., Xin, M.D.: Heat Transfer Enhancement with Metal Powder on the Inner Wall of a Certain Two Phase Closed Thermosyphon. Journal of Chongqing University, 32–37 (1984)
11. Ghajar, M., Darabi, J.: Numerical Modeling of Evaporator Surface Temperature of a Micro Loop Heat Pipe at Steady State Condition. Journal of Micromechanics and Microengineering 15, 1963–1971 (2005)
12. Chen, M.M., Faghri, A.: An Analysis of The Vapor Flow and The Heat Conduction Through the Liquid-wick and Pipe Wall In a Heat Pipe With Single or Multiple Heat Sources. International Journal of Heat and Mass Transfer 33, 1945–1955 (1990)
13. Schmalhofer, J., Faghri, A.: A Study of Circumferentially Heated and Block-heated Heat Pipes-II. Three-dimensional Modeling as a Conjugate Problem. International Journal of Heat and Mass Transfer 36, 213–226 (1993)
14. Zhu, N., Vafai, K.: Analysis of Cylindrical Heat Pipes Incorporating the Effects of Liquid-vapor Coupling and Non-Darcian Transport - a Closed Form Solution. International Journal of Heat and Mass Transfer 42, 3405–3418 (1999)
15. Chernysheva, M.A., Maydanik, Y.F.: Numerical Simulation of Transient Heat and Mass Transfer in a Cylindrical Evaporator of a Loop Heat Pipe. International Journal of Heat and Mass Transfer 51, 4204–4215 (2008)

Evaluation of Micronozzle Performance through DSMC, Navier-Stokes and Coupled DSMC/Navier-Stokes Approaches

Federico La Torre[1], Sasa Kenjeres[1], Chris R. Kleijn[1], and Jean-Luc P. A. Moerel[2]

[1] Department of Multi Scale Physics, Faculty of Applied Sciences, Delft University of Technology,
Prins Bernhardlaan 6, 2628 BW Delft, The Netherlands
F.LaTorre@tudelft.nl, S.Kenjeres@tudelft.nl,
C.R.Kleijn@tudelft.nl
[2] Department of System Performance and Survivability, TNO Defence, Security and Safety,
P.O.Box 45,2280 AA Rijswijk, The Netherlands
Jean-Luc.Moerel@tno.nl

Abstract. Both the particle based Direct Simulation Monte Carlo (DSMC) method and a compressible Navier-Stokes based continuum method are used to investigate the flow inside micronozzles and to predict the performance of such devices. For the Navier-Stokes approach, both slip and no-slip boundary conditions are applied. Moreover, the two methods have been coupled to be used together in a hybrid particle-continuum approach: the continuum domain was then investigated by solving the Navier-Stokes equations with slip wall boundary condition, whereas the region of rarefied regime was studied by DSMC. The section where the domain was split was shown to have a great influence in the prediction of the nozzle performance.

Keywords: Direct Simulation Monte Carlo, Coupled Method, Rarefied Gas Flow, Slip Regime, Micronozzle.

1 Introduction

Various trends in the spacecraft industry are driving the development of so-called micro-satellites (which have a typical volume of 1 dm^3). For missions that require a satellite propulsion system, the existing propulsion systems are too large and too heavy. To address this problem, in The Netherlands a large multidisciplinary research program is being carried out [1] aimed at the development of micro-propulsion systems with a typical thrust in the order of mN. Such propulsion systems can be used to maintain or adjust the orbit of space micro-satellites or to provide long duration low thrust acceleration.

One of the simplest forms of micropropulsion is the cold gas thruster: the gas, pressurized in a microtank, is accelerated and expanded through a convergent-divergent nozzle. The typical dimensions of the throat of the nozzles are below one millimeter. Due to these small characteristic dimensions, there are some important differences in the gas flow field in such a micro-nozzle, compared to conventional large scale nozzles. The increase of the surface-to-volume ratio and the reduction of the Reynolds number lead to the presence of large viscous losses and to the formation of thicker boundary layers (relative to the geometrical dimensions), with a consequent lower efficiency of such systems. Also the effect of surface roughness, with a characteristic length comparable to the throat diameter, cannot be neglected. These aspects of micro-thruster flow have been investigated in [2].

Another important aspect of micro-nozzle gas flow, especially when going even further down in thrust to the μN range, is the effect of rarefaction. At these low thrusts and small nozzle dimensions, the throat diameter and gas flow boundary layer thickness become comparable to mean free path of the molecules and we enter a different gas flow regime. An important parameter in this respect is the Knudsen number, defined as the ratio between the mean free path of the gas molecules λ, and a characteristic length L of the flow:

$$\mathrm{Kn} = \frac{\lambda}{L} \tag{1}$$

When Kn > 10, molecule-molecule interactions are negligible compared to molecule-wall interactions, and the flow is in the so-called free molecular regime. This regime can be accurately and efficiently modeled through free-flight (ballistic) molecular models. When Kn < 0.1, the flow is dominated by intermolecular interactions and can be described by continuum equations, such as the Navier-Stokes equations. However, for 0.1 > Kn > 0.01 important deviations from the Navier-Stokes solutions occur in the vicinity of solid walls. This is the so-called slip flow regime, requiring the use of modified boundary conditions for momentum and energy transfer at the walls. For 10 > Kn > 0.1, molecule-molecule and molecule-wall interactions are equally important, and the flow is in the so-called transition regime. In this regime, neither continuum models nor free molecular models can be applied.

When operated in space, the gas flow in micro-nozzles experiences all regimes mentioned above, from the continuum regime (in the gas chamber and the convergent part of the nozzle), to slip flow and transition regimes (in the divergent part of the nozzle), up to the free molecular regime (far from the exit of the nozzle). The need to accurately and consistently model all these regimes is a challenge from a computational point of view.

Some attempts have been made to use Navier-Stokes equations with slip wall boundary conditions, for example in [3] [4] [5] [6]. However, such an approach is not accurate when the Knudsen number exceeds 0.1, see e.g. [16] [17]. In the transition regime, the most accurate and flexible method to use is Direct Simulation Monte Carlo (DSMC) [22]. It is more efficient than deterministic molecular models such as Molecular Dynamics (MD), and more flexible and generally applicable than methods based on direct solution of the Boltzmann equations. Full DSMC simulations, for different micronozzle configurations were performed in [7] [8] [9] [10].

In principle, DSMC is valid for all flow regimes. However, in order for its solutions to be accurate the requirements on number of modeled particles, grid size and time step size increase severely with decreasing Knudsen number. As a result, the computational expenses of a proper DSMC simulation scale with Kn^{-4}. In practice it is therefore not possible to use DSMC in all flow regions, and the application of DSMC to an entire micro-nozzle flow cannot be expected to be accurate in the low Kn number zones.

In order to overcome this problem, a hybrid Navier-Stokes / DSMC model can be considered. The method applies the continuum approach in low Kn regions, while DSMC is applied in regions where rarefaction effects need to be considered. Some of these hybrid models were proposed in [11] [12] [13] [14] [15].

In the present paper, the performance of a μN thrust micro-nozzle is evaluated through different numerical models, i.e. Navier-Stokes equations with slip flow boundary conditions, full DSMC, and coupled Navier-Stokes / DSMC. After a brief introduction on the studied problem (Section 2), the coupled hybrid approach will be explained in the Section 3, while the numerical results will be shown in Section 4. Conclusions and some remarks will be addressed in Section 5.

2 Description of the Problem

The studied nozzle is shown in Fig. 1. This geometry is typical for micronozzle thrusters as used in practice [1] [2]. Two different regions can be distinguished: a convergent part, where the flow is accelerated from the initial subsonic velocity to the sonic condition (which is reached at the throat), and a divergent part, where the flow expands to the outlet in a supersonic regime.

The analyzed configuration has both convergent and divergent angles equal to 20°, an inlet area equal to 7.27 times the throat area, and an outflow area of 23.23 times the throat area. The throat radius is 3.56 μm, while a sharp angle is considered to connect the convergent and the divergent. In the inlet, nitrogen gas is introduced at a fixed pressure and temperature of respectively P_0=3.5 bar and T_0=300 K. A vacuum condition is assumed outside. The walls are modeled as isothermal at T_0=300 K. Under these conditions, the ideal thrust T_{ideal} computed from the 1D isentropic flow theory [18], is about 25 μN.

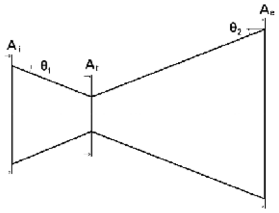


Fig. 1. The nozzle geometry configuration considered in the present study, with the geometric parameters of interest

3 Numerical Approach

3.1 The CFD Solver

All the CFD simulations presented in this paper have been obtained with the general purpose code Fluent version 6.3 [19]. The studied nozzle has a conical shape and therefore a 2D-axisymmetric flow is considered. The grid, generated with Gambit [20], consists of quadrilateral elements, with 100 grid cells in the radial direction and about 600 grid cells in the axial direction. The nitrogen gas is treated as an ideal gas, with temperature dependent viscosity and thermal conductivity computed from kinetic theory. As an initial guess for the solution, the pressure P_e at the outflow was computed from the 1D isentropic flow theory [18].

The solved equations include the continuity equation, the momentum balance (Navier Stokes) equations in compressible form, and the thermal energy equation. The equations were discretized in space using a second order upwind scheme, and solved using Fluent's explicit coupled pressure based solver. This choice was done in order to enable the option to impose a slip velocity boundary condition at the wall, according to the formulas by Maxwell and von Smoluchowski [21], approximated by the solver as explained in [19].

3.2 Direct Simulation Monte Carlo

The DSMC method, as developed originally by G.A. Bird, is comprehensively described in [22]. This method is not based on solving partial differential equation, such as the Navier-Stokes equations, but describes the state of the system by computing the positions and velocities of computational particles, each of which represents an assigned number of particles in the real flow. The computational domain is divided into grid cells, through which the particles can move. All the properties of the particles are stored and updated each time step during the calculation. First, the particles are moved as if they did not interact, and any particles that reach a boundary are processed according to the appropriate boundary condition. Second, after all the particles have moved, a certain number of particles from each grid cell are randomly selected for collisions, according to one of the collisions models [22]. This decoupling between the translational movement of the particles and their collisions is accurate only when the time step is a fraction of the mean collision time. On the other hand, the random selection of the particles as possible candidates to be collision partners leads to the restriction that their mean separation needs to be a fraction of the mean free path. This means that the cell size should be small enough in order to avoid that energy or momentum are transported over long distances in an unrealistically short time. Finally, the number of particles per cell needs to be high enough to avoid repeated collisions between the same particles and make the collision process statistically accurate. All these constraints make DSMC computationally expensive.

The DSMC solver used in the present study has been implemented in the 3-dimensional general purpose DSMC code X-Stream [24] and validated for a wide range of problems [25], amongst which the case of a low pressure micronozzle as studied in [9]. In all the simulations presented in this paper, a Variable Soft Sphere (VSS) model has been used, with the corresponding parameter set from the literature [22]. For the interaction between the nitrogen particles and the isothermal silicon walls a thermal accommodation coefficient of 0.5 was assumed, according to the formula provided in [23]. In order to model axial-symmetry, a radial weighting factor has been introduced, as described in [22] [25], such that a molecule located far from the axis represents more real molecules than one near the axis.

3.3 The Hybrid CFD / DSMC Approach

Figure 2 shows a first comparison between the solutions obtained from a full CFD simulation, both with a partial slip (wall accommodation coefficient 0.5) and a no-slip wall boundary condition, and the solutions from full DSMC. When assuming partial slip at the wall, the CFD solution matches the DSMC solution much better, particularly in the divergent. However, it cannot be expected that the full DSMC solution accurately represents the 'true' flow physics, since the DSMC computational requirements, as explained above, could not be satisfied in the whole domain, in particular in the convergent part of the nozzle where the pressure is high and the Knudsen number is low.

Because of the above, we will develop a hybrid approach, applying CFD with partial slip in the upstream (convergent) part of the domain, and DSMC in the downstream (divergent) part of the domain.

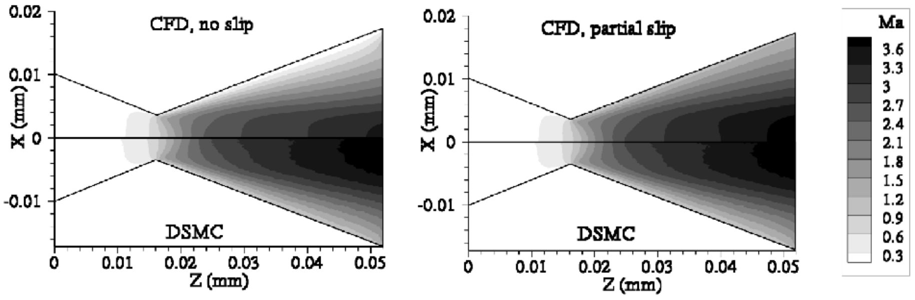


Fig. 2. Comparison of Mach number contours from the solution obtained by full CFD (half upper picture) and full DSMC (half bottom picture). On the left, a no-slip velocity was imposed at the wall for the CFD problem, while on the right a slip flow boundary condition with accomodation coefficient 0.5 was applied.

In the present paper a single step coupling approach between CFD and DSMC has been conducted. First, a CFD simulation is performed for the entire nozzle. From this an estimation of the local Knudsen number throughout the nozzle is made, see Fig. 3. Four different definitions for Kn have been considered, depending on the definition of the characteristic length L in (1).

If L is taken to be the throat diameter D_t, (1) becomes:

$$Kn = \frac{\lambda}{D_t} \qquad (2)$$

To account for local variations in characteristic length scales of the flow, a more accurate definition of Kn can be obtained using the gradient length scale of a certain physical quantity:

$$Kn_Q = \frac{\lambda}{Q}|\nabla Q| \qquad (3)$$

Figure 3 shows Knudsen numbers based on (a) the throat diameter, and gradient length scales of (b) pressure, (c) density and (d) temperature. When the maximum of the four computed values of Kn exceeds a value of approximately 0.1, the continuum CFD approach is considered to be inaccurate. From Fig. 3 it is clear that this is the case for (a large part of) the divergent.

Therefore, the computational domain was subsequently split between a CFD part upstream, and a DSMC part downstream of a certain vertical division plane. The predicted solution from CFD at this division plane was then applied as inlet boundary condition for a DSMC solution downstream from the division plane. We considered division planes located at the throat, and 6, 12, 18 and 24 μm downstream the throat; these sections will be referred, respectively, as Th, L1, L2, L3 and L4.

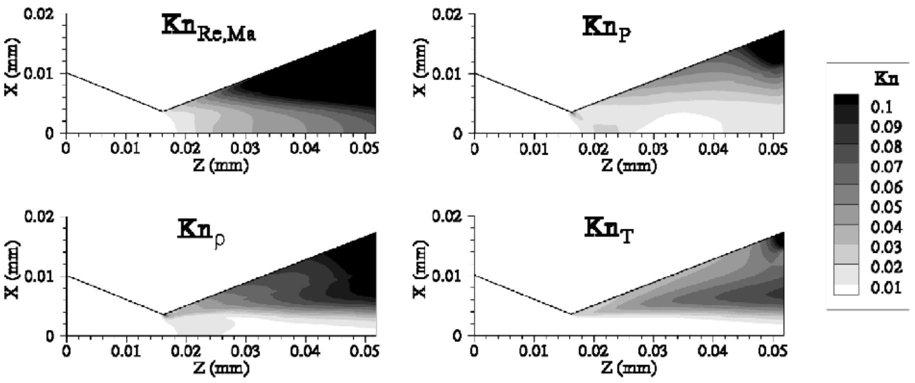


Fig. 3. Knudsen number contours evaluated by (2) (upper left picture), and (3) with pressure (upper right picture), density (bottom left picture) and temperature (bottom right picture) as characteristic quantity, from the solution obtained by full CFD with no slip wall boundary condition

4 Results

To compare the full CFD, full DSMC and hybrid CFD/DSMC approaches for modeling the gas flow in the micro nozzle, we will focus on the predicted nozzle performance. This performance is evaluated through two parameters. The first is defined as:

$$\eta = \frac{T_{real}}{T_{ideal}} \tag{4}$$

where the ideal thrust T_{ideal} is given by the 1D isentropic flow theory, and the real thrust T_{real} is computed from

$$T_{real} = \int_{\substack{\text{outlet}\\ \text{area}}} \left[m V_e + \left(p_e - p_a \right) \right] dA \tag{5}$$

The subscript e refers to the conditions at the exit section of the nozzle, while the subscript a refers to the ambient conditions outside the nozzle.

The second performance parameter is the specific impulse, defined as:

$$I_{sp} = \frac{T}{\dot{m}\, g} \tag{6}$$

which describes the change in momentum of the satellite per unit of propellant weight (on earth), so that the higher it is, the less propellant is needed to gain a certain amount of momentum.

The nozzle performances as computed from the different modeling approaches have been listed in Table 1.

Table 1. Nozzle performance evaluated by the solutions from different numerical approaches

Configuration	T, mN	η	I_{sp}, s
full CFD, no-slip wall	1.96×10^{-2}	84.52	68.44
full CFD, partial slip wall	2.06×10^{-2}	88.78	68.98
full DSMC	2.00×10^{-2}	86.26	66.71
CFD slip, DSMC from Th	1.99×10^{-2}	86.04	66.85
CFD slip, DSMC from L1	2.04×10^{-2}	87.83	68.02
CFD slip, DSMC from L2	2.05×10^{-2}	88.29	68.37
CFD slip, DSMC from L3	2.06×10^{-2}	88.97	68.90
CFD slip, DSMC from L4	2.07×10^{-2}	89.33	69.17

When a slip wall boundary condition is applied, the full CFD approach predicts a higher performance than the full DSMC approach both in terms of thrust and specific impulse. The presence of slip velocity allows the flow to reach higher velocities at the outflow. In general, as can be noted from Fig. 2, the use of slip flow boundary conditions leads to a decrease of the subsonic part of the boundary layer in the divergent, and to an overall decrease of the thickness of that boundary layer. This leads to a larger expansion of the flow, with a consequent higher velocity and a lower pressure in the outlet. This can be observed through the investigation of the exit profiles for velocity and pressure, as shown in Fig. 5.

On the other hand, a full CFD simulation with no-slip wall boundary condition gives an under-prediction of the nozzle thrust compared to full DSMC. This lower thrust can be related to the presence of a thicker subsonic boundary layer and to a consequent lower mass flow and no velocity close to the wall. In contrast with the underprediction of the thrust, the full CFD simulation with no-slip boundary condition leads to an overprediction of the specific impulse compared to the full DSMC solution, because the mass flow in this case is lower than the one observed when a partial slip wall is modeled. In general, both the full CFD and the hybrid CFD/DSMC solutions led to a higher specific impulse than the full DSMC solution.

From Table 1 it can be seen that the thrusts and specific impulses predicted by the hybrid CFD/DSMC modeling approach, depend on the location of the vertical division plane between the CFD and DSMC regions. When the division plane is located at the throat, the thrust and specific impulse predicted by the hybrid solution are similar to those of a full DSMC solution. When the division plane is located far downstream in the divergent, the thrust and specific impulse predicted by the hybrid approach are similar to that of a full CFD solution. This because a DSMC solution started too far downstream from the throat does not properly take into account the whole domain where rarefied effects are present and is not able to simulate the full expansion of the flow, with a consequent too high exit pressure (5). The solution from the hybrid approach, both in terms of efficiency and of the flow field inside the nozzle, approaches that of a full DSMC approach when most of the divergent is solved with DSMC. This is also visible in Fig. 4.

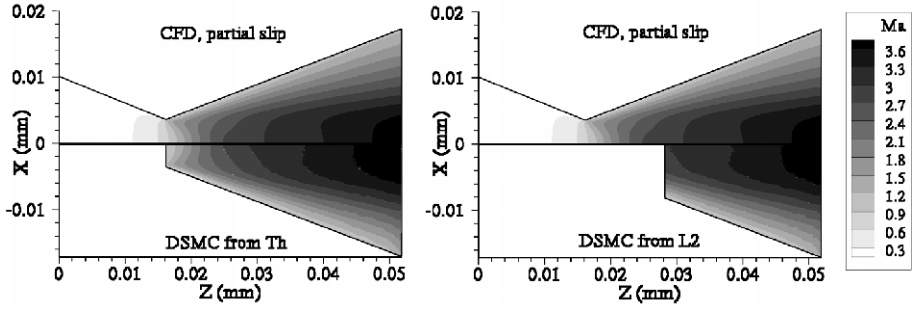


Fig. 4. Comparison of Mach number contours from the solution obtained by full CFD and partial slip wall velocity (half upper picture) and DSMC with inlet boundary conditions extrapolated from the CFD solution (half bottom picture), at the throat (left), section L2 (right)

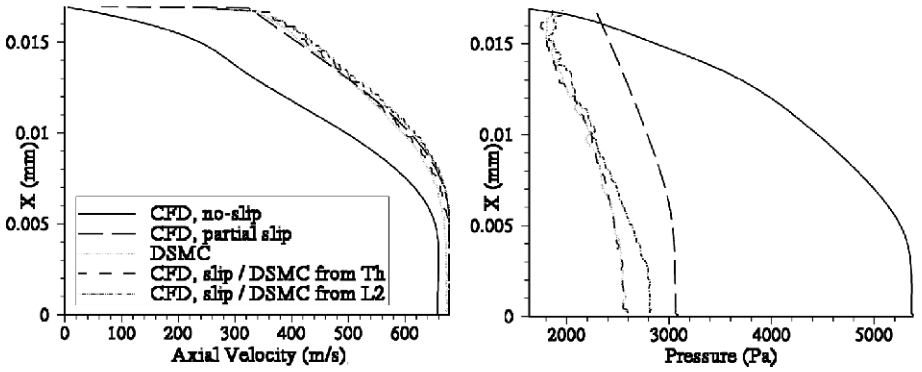


Fig. 5. Profiles for the exit velocity (left) and exit pressure (right) obtained by CFD, DSMC and coupled CFD/DSMC

5 Conclusions

A fixed micronozzle configuration allowing a thrust of about 25 μN has been studied through CFD, DSMC and coupled CFD/DSMC simulations. The continuum model, both with no-slip and slip flow at the walls, showed different results from DSMC for the flow field inside the nozzle in the divergent, especially close to the outflow, where strong rarefied effects start to appear. In general, a better agreement between DSMC and CFD was observed when a slip wall boundary condition was implemented. From this point of view, a fully CFD simulation with slip wall could be considered accurate enough to predict the flow field inside a 25 μN thruster.

On the other hand, although the outflow velocity in the expansion core of the nozzle was close to the solution from DSMC, a full CFD solution with slip boundary condition was found to overpredict the thrust compared to a DSMC solution. It was found that this is due to the fact that a CFD solution – even with slip wall boundary conditions – overpredicts the pressure field in the outflow region, which enhances the nozzle performance, both in terms of efficiency and of specific impulse. A static hybrid CFD / DSMC approach confirmed the importance of the pressure term in the calculation of the thrust of the nozzle, and showed that a great part of the divergent needs to be modeled by DSMC to allow the prediction of a full expansion. In this case, the final solution strongly depends on the division plane downstream from which DSMC is applied, and approaches to the full DSMC solution when a larger part of the divergent is solved through DSMC.

Since the present, static and one-way, coupling between CFD and DSMC proved to be rather sensitive to the exact location of the division plane between the two computational regions, there is a need to develop an adaptive, two-way coupling approach, in which the results from the DSMC solution in the downstream part of the nozzle are used to update the boundary conditions for the CFD solution in the upstream part, and an adaptive overlap region is applied to ensure that the CFD and DSMC solutions are consistent with each other. We believe that such an approach can be realized using the coupling between a general purpose CFD code such as Fluent, and a DSMC code such as X-Stream. It should be pointed that numerical issues in the interface between the two methods could also play a role. Since the difference in the performance as predicted by the various approaches is less than a few percents, extra cases with even lower thrusts and nozzle dimensions should be studied to conclusively determine the merits of the various simulation approaches for micro nozzle gas flow. This is object of current investigation.

Acknowledgement

This project is founded by MicroNed, one of the Dutch BSIK research programs.

References

1. Moerel, J.L.P.A., Sanders, H.M., Louwerse, M.C., Jansen, H.V., Boscher, J., Zandbergen, B.T.C., La Torre, F., Kenjeres, S., Kleijn, C.R.: Development of Micro Propulsion System Technologies for Minisatellites in The Netherlands. In: Proceedings of the 5th Int. Spacecraft Propulsion Conference (2008)
2. La Torre, F., Kenjeres, S., Kleijn, C.R., Moerel, J.L.P.A., Zandbergen, B.T.C.: Influence of Boundary Layer Formation and Surface Roughness on the Thrust of Micro-Nozzle. In: Proceedings of the 5th Int. Spacecraft Propulsion Conference (2008)

3. Liu, M., Zhang, X., Zhang, G., Chen, Y.: Study on Micronozzle Flow and Propulsion Performance using DSMC and Continuum Methods. Acta Mechanica Sinica 22, 409–416 (2006)
4. Markelov, G.N., Ivanov, M.S., Ketsdever, A.D., Wadsworth, D.C.: Numerical Study of Cold Gas Micronozzle Flows. In: Proceedings of the 33rd Aerospace Science Meeting and Exhibit, 1999, AIAA 99-0166 (1999)
5. Gadepalli, V.V.V., Lin, C.: Navier-Stokes Modeling of Gas Flows in a De-Laval Micronozzle. In: Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit (2006)
6. Mo, H., Lin, C., Gokaltun, S., Skudarnov, P.V.: Numerical Study of Axisymmetric Gas Flow in Conical Micronozzle by DSMC and Continuum Methods. In: Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit, AIAA 2006-991(2006)
7. Xie, C.: Characteristics of Micronozzle Gas Flows. Journal of Physics of Fluids 19(3) (2007)
8. Markelov, G.N., Ivanov, M.S.: Numerical Study of 2D/3D Micronozzle Flows. In: Proceedings of the 22nd International Rarefied Gas Dynamics Symposium, AIP Conference Proceedings, vol. 585, pp. 539–546 (2001)
9. Buoni, M., Kietz, D., Aslam, K., Subramaniam, V.V.: Simulation of a Compressible Gas Flow in a Mmicronozzle. In: Proceedings of the 35th AIAA Thermophysics Conference, AIAA 2001-3073 (2001)
10. Alexenko, A.A., Levin, D.A., Gimelschin, S.F., Collins, R.J.: Numerical Modeling of Axisymmetric and Three-Dimensional Flows in MEMS Nozzle. AIAA Journal 40(5), 897–904 (2002)
11. Garcia, A., Bell, J.B., Crutchfield, W.Y., Alder, B.J.: Adaptive Mesh and Algorithm Refinement using Direct Simulation Monte Carlo. Journal of Computational Physics 154, 134–155 (1999)
12. Lou, T., Dahlby, D.C., Baganoff, D.: A Numerical Study Comparing Kinetic Flux-Vector Splitting for the Navier-Stokes Equations with a Particle Method. Journal of Computational Physics 145, 489–510 (1998)
13. Le Tallec, P., Mallinger, F.: Coupling Boltzmann and Navier-Stokes Equations by Half Fluxes. Journal of Computational Physics 136 (1997)
14. Wijesinghe, H.S., Hadijconstantinou, N.G.: Discussion of Hybrid Atomistic-Continuum Methods for Multiscale Hydrodynamics. International Journal for Multiscale Computational Engineering 2, 189–202 (2004)
15. Abbate, G., Thijsse, B.J., Kleijn, C.R.: Validation of a Hybrid N-S/DSMC Method for Multi-Scale Transient and Steady-State Gas Flows. International Journal for Multiscale Computational Engineering 6, 1–12 (2008)
16. Cercignani, C.: The Boltzmann Equation and its applications. Springer, Heidelberg (1988)
17. Hadjiconstantinou, N.G.: Comment on Cercignani's Second-Order Slip Coefficient. Physics of Fluids 15, 2352 (2003)
18. Sutton, G.P., Biblarz, O.: Rocket Propulsion Elements. John Wiley & Sons, Chichester (2001)
19. Fluent 6 Tutorial Guide. Fluent Incorporate (2005)
20. Gambit 2 Tutorial Guide. Fluent Incorporate (2001)
21. Gad-el-Hak, M.: The Fluid Mechanics of Microdevices – the Freeman Scholar Lecture. Journal of Fluids Engineering 121, 5–33 (1999)
22. Bird, G.A.: Molecular Gas Dynamics and the Direct Simulation of Gas Flows. Oxford Science Publications, Oxford (1994)
23. Goodman, F.O., Wachman, H.Y.: Formula for thermal accommodation coefficients. Journal of Chemical Physics 46(6), 2376–2386 (1967)
24. CVD-X User Manual. TNO (2007)
25. Dorsman, R.: Numerical Simulations of Rarefied Gas Flows in Thin Film Processes. PhD thesis, Delft (2007)

A Multilevel Multiscale Mimetic (M^3) Method for an Anisotropic Infiltration Problem

Konstantin Lipnikov, David Moulton, and Daniil Svyatskiy

Los Alamos National Laboratory, Theoretical Division,
Los Alamos, NM 87545, USA
`{lipnikov,moulton,dasvyat}@lanl.gov`

Abstract. Modeling of multiphase flow and transport in highly heterogeneous porous media must capture a broad range of coupled spatial and temporal scales. Recently, a hierarchical approach dubbed the Multilevel Multiscale Mimetic (M^3) method, was developed to simulate two-phase flow in porous media. The M^3 method is locally mass conserving at all levels in its hierarchy, it supports unstructured polygonal grids and full tensor permeabilities, and it can achieve large coarsening factors. In this work we consider infiltration of water into a two-dimensional layered medium. The grid is aligned with the layers but not the coordinate axes. We demonstrate that with an efficient temporal updating strategy for the coarsening parameters, fine-scale accuracy of prominent features in the flow is maintained by the M^3 method.

Keywords: multiscale, hierarchical, two-phase flow, heterogeneous porous media, infiltration.

1 Introduction

High fidelity simulations of multiphase flow and transport in porous media play a key role in driving scientific advances in a broad range of complex multiscale applications, including carbon sequestration, aquifer assessment and protection, and nuclear waste disposal. However, there are fundamental mathematical and computational hurdles that arise from the wide range of strongly coupled spatial and temporal scales. For example, the permeability of porous media is highly heterogeneous and may span several orders of magnitude, from nearly impermeable barriers to high-permeable flow channels. To address this challenge we have developed the new Multilevel Multiscale Mimetic (M^3) method [1].

The M^3 method builds recursively a problem-dependent *multilevel hierarchy of models*. Each model preserves important physical properties of the continuum model, such as *local mass conservation*. In contrast to two-level methods, such as the multiscale finite element methods [2,3,4], the multiscale mortar finite element method [5] and the Multiscale Finite Volume (MSFV) method [6], the multilevel hierarchy facilitates large total coarsening factors, of 100 or more in each coordinate direction.

Building and maintaining the hierarchy of models incurs only a moderate computational overhead especially for a small coarsening factor on each level. The M^3 method supports unstructured polygonal meshes, and is readily extended to unstructured polyhedral meshes. In addition, it can handle full permeability tensors and accommodates general coarsening strategies to capture accurately the complexity of the heterogeneous subsurface environment.

The M^3 method [1] merges two computational strategies to balance accuracy and efficiency in two-phase flow simulations. The first strategy is the algebraic coarsening developed by Y. Kuznetsov for single phase flows that reduces the degrees of freedom inside coarse-grid cells [7]. The second strategy is a *novel* approach to the conservative coarsening of velocities on the edges of a coarse-grid cell. This combination ensures that the coarse-scale system has the same sparsity structure as the fine-scale system, and with recursion leads to a multilevel algorithm. Due to its algebraic nature, the method can be adapted to other discretizations of the same algebraic form, such as the mixed finite element and finite volume methods.

In this work we demonstrate the capabilities of the M^3 method for infiltration into a stratified porous medium that is composed of sloping and non-uniform layers and strongly anisotropic inclusions. The majority of two-level methods solve localized fine-scale flow problems to capture the influence of fine-scale structures. Formulation of local problems with appropriate localized sources and boundary conditions can be problematic, particularly for anisotropic media [8]. The multiscale mortar finite element method [5] offers a more robust approach for this case, as the accuracy can be controlled via refinement of the mortar space. The M^3 method uses a different mechanism for controlling the accuracy via efficient incorporation of global information into the hierarchy of models. Numerical experiments demonstrate that the M^3 method can maintain fine-scale solution accuracy, even with large coarsening factors, while significantly reducing the overall simulation time.

The paper outline is as follows. In Section 2, we present the mathematical model of the infiltration problem. In Section 3, we describe essential features of the M^3 method. In Section 4, we illustrate the effectiveness of the method with the simulation of water infiltration into a layered anisotropic porous medium.

2 Mathematical Model of a Two-Phase Flow

We consider the flow of two immiscible phases, non-wetting (e.g., air, denoted a) and wetting (e.g., water, denoted w), in a two-dimensional domain subject to gravity (see, e.g., [9,10]). The effects of compressibility and capillary pressure are neglected. Conservation of each phase implies,

$$\phi \frac{\partial S_j}{\partial t} + \nabla \cdot \boldsymbol{u}_j = -q_j, \qquad j = a, w\,, \tag{1}$$

where S_j and $\boldsymbol{u}_j$ are the saturation and the velocity of phase j, respectively, and the porosity, ϕ, is assumed constant. The Darcy velocity of phase j is given by,

$$\boldsymbol{u}_j = -\mathbb{K} \cdot \lambda_j (\nabla p_j - \rho_j \mathbf{g})\,, \tag{2}$$

where $\mathbb{K}$ is the absolute permeability tensor; $\lambda_j = k_{rj}/\mu_j$ is the relative mobility of phase j, with k_{rj} the relative permeability and μ_j the viscosity; ρ_j is the density and $\mathbf{g}$ is the gravitational acceleration vector.

Assuming that the two phases fill the pore volume we have the constraint $S_a + S_w = 1$. This may be combined with the sum over j of (1) to yield the bulk fluid conservation law,

$$\nabla \cdot \boldsymbol{u} = q_a + q_w \,. \tag{3}$$

Here the total or bulk fluid velocity, $\boldsymbol{u} = \boldsymbol{u}_a + \boldsymbol{u}_w$, may be written in the form,

$$\boldsymbol{u} = -\mathbb{K}(\lambda \nabla p - (\lambda_a \rho_a + \lambda_w \rho_w)\mathbf{g}), \tag{4}$$

where $\lambda = \lambda_w + \lambda_a$ is the total mobility, and neglecting capillary pressure we have set $p = p_a = p_w$ to be the reference pressure. Thus, the two-phase system that we model is comprised of an elliptic equation for the pressure, obtained by substitution of (4) into (3), and a hyperbolic equation for the water saturation, $S = S_w$, given by (1) with $j = w$. The initial and boundary conditions used to close the model are given in Section 4.

3 Multiscale IMPES Method

3.1 Time Integration and Fine-Scale Discretization

We use the IMPES (IMplicit Pressure and Explicit Saturation) time integration method. First, the pressure equation is solved to define the velocity field. Second, the hyperbolic equation for the water saturation is integrated explicitly using the single-point upwind finite volume method [11].

To discretize the pressure we consider a polygonal partition of the domain Ω, denoted Ω_h, that is the union of the polygonal cells e_i, $i = 1, \ldots, N$. We apply the Mimetic Finite Difference (MFD) method [12] to the first-order form of the pressure equation, given by (4) and (3). For each cell e_i, we define one pressure unknown, p_i, which represents the integral average of p. Similarly, for each edge ℓ_j , we define one unknown, u_j, which represents the average normal flux $\boldsymbol{u} \cdot \boldsymbol{n}$ (a scalar) through this edge. The MFD discretization of the pressure equation reads

$$\begin{bmatrix} \mathbf{M}(\mathbf{S}^n) & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_g^n \\ \mathbf{q}^n \end{bmatrix} . \tag{5}$$

Here n denotes the time step, $\mathbf{M}(\mathbf{S}^n)$ is the mass matrix computed using the current saturation and $\mathbf{B}$ is the discrete divergence operator. The source $\mathbf{q}_g^n$ captures Dirichlet data and the gravitational term in (4), and $\mathbf{q}^n$ captures Neumann data and the source/sink term in (3).

3.2 Two-level Method for the Pressure Equation

We begin by describing a two-level coarsening method, a building block for the multilevel method. Let $N_0 = N$, $N_1 \leq cN_0$, with $0 < c < 1$, and

$$\Omega_H = \bigcup_{I=1}^{N_1} E_I, \qquad E_I = \bigcup_{i \in \mathcal{F}(E_I)} e_i,$$

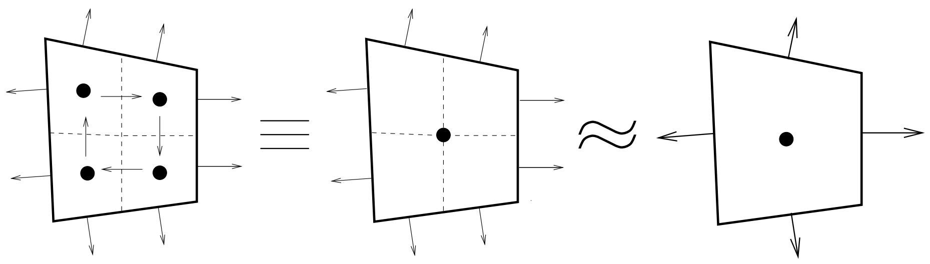


Fig. 1. Schematic of the two steps of the two-level upscaling method for a 2×2 square macro-cell. The cell-centered pressure unknowns are represented by circles, and the velocity unknowns are represented by arrows. The first step is an equivalent reduction, while the second step is approximate.

where $\mathcal{F}(E_I)$ is a set of indices of fine-grid cells forming the *macro-cell* E_I. We assume that the coarse-grid partition Ω_H is non-overlapping and conformal. Note that each *macro-edge* of a macro-cell E_I is not necessarily a straight line, as it is simply a collection of fine-grid edges.

The two-level method consists of two steps, which are illustrated in Fig. 1. For simplicity, we omit the time superscript n, and consider aggregation of four cells into a single macro-cell. First, for each macro-cell, we eliminate all internal flux unknowns, and replace all internal pressure unknowns with a single pressure [7]. By construction, this single pressure on each macro-cell is the volume-weighted average of the corresponding fine-grid pressures:

$$P_I = \frac{\sum\limits_{i \in \mathcal{F}(E_I)} p_i |e_i|}{\sum\limits_{i \in \mathcal{F}(E_I)} |e_i|}.$$

This first step is an *equivalent* modification of the original system (see [1] for details). The unknowns in the resulting system are shown in Fig. 1(middle).

The second step is to perform a *conservative* flux coarsening procedure that leaves one flux unknown U_J per macro-edge L_J. Mass conservation dictates that the coarse flux on a macro-edge must be a weighted sum of the corresponding fine-grid fluxes. This coarsening makes the final structure of the reduced system, shown in Fig. 1(right), identical to the original system. For each fine-grid flux u_j we introduce the coarsening parameter α_j, as follows

$$u_j = \alpha_j U_J. \tag{6}$$

This parameter is related to the first moment of the flux, and plays an important role in the M^3 algorithm. It is used to define the interpolation of the flux, and it influences the coarse-scale pressure model. Suppose that the problem (5) is solved exactly. Then we can calculate coarse-grid fluxes U_J and derive coarsening parameters α_j. Such parameters will make the upscaling procedure exact. However, these parameters are functions of time, and accurately recomputing them at each time step would be equivalent in cost to the fine-scale IMPES formulation. In the M^3 method these parameters are derived from an approximate solution of the fine-scale problem (5) and we recompute these parameters only when it is necessary to maintain accuracy (see Section 3.3).

3.3 Multilevel Hierarchy of Pressure Models

The two-level coarsening method results in coarse-grid equations that have the same structure as the fine-grid equations. Hence a multilevel algorithm follows naturally by applying this process recursively, and generates a hierarchy of discrete upscaled pressure models. To maintain the accuracy of this hierarchy without compromising the overall performance of the method, efficient update strategies are required. We have implemented the local and global update strategies described below.

Local Update Strategy. Here, the M^3 method is focused on the total mobility. The total mobility changes significantly in the vicinity of a sharp water front. The M^3 method updates cell-based matrices (over the whole hierarchy) that are affected by these changes. More precisely, we use the criterion proposed in [13],

$$\frac{1}{1+\varepsilon_\lambda} \leq \frac{\lambda^n}{\lambda^{n-1}} \leq 1+\varepsilon_\lambda,$$

where ε_λ is the user defined threshold, usually 0.1. If this condition is violated, the cell-based matrices are updated.

Global Update Strategy. In simulations of flow through porous media with long correlation lengths, or strong anisotropy, local updates of the hierarchy are not sufficient to maintain the accuracy. In these cases, the elliptic nature of the pressure equation is accentuated and global information is needed. Therefore, we recalculate flux coarsening parameters during the simulation. These updates are equi-distributed in time approximately every few hundred time steps. The impact of these updates on the accuracy of the simulation is discussed in Section 4.

To maintain the overall performance of the simulation, an efficient global approximation strategy for the α's is needed. In [1] we proposed to leverage the strength of a robust variational multigrid method for this purpose. The saddle-point problem (5) is hybridized and reduces to a symmetric positive-definite problem for the Lagrange multipliers. We use preconditioned conjugate gradient iterations with Ruge-Stüben algebraic multigrid [14] as the preconditioner, denoted PCG(AMG), to solve approximately for the Lagrange multipliers. The convergence tolerance ε_r is usually set to $10^{-2} - 10^{-3}$, and is reached in only a few (3-10) iterations. This approach captures critical changes in global flow behavior. In our original study [1] we demonstrated its effectiveness for highly heterogeneous porous media with long correlation lengths. Here we focus on the challenging problem of infiltration into a strongly anisotropic medium.

4 Numerical Experiments

In our numerical experiments we model the infiltration of water into an unsaturated two-dimensional porous medium, shown schematically in Fig. 2(a). Initially the pore space is filled with air and a small residual water saturation. The medium is composed of four sloping layers and two elongated inclusions, and is partly motivated by the study in [15]. The interfaces of these features are not aligned with the coordinate axes. A structured orthogonal mesh is mapped onto these features so that each mesh cell has homogeneous properties. A 32×32 mesh is shown in Fig. 2(b), while the mesh used in our numerical experiments is 128×128. Based on this mapping, we define a local coordinate system at the centroid of each mesh cell that is aligned with the associated layer or inclusion. In this local coordinate system the absolute permeability tensor is assumed to be diagonal. The diagonal permeability tensors, as well as their anisotropy ratios, are presented in Table 1. Note that since the local coordinate axes are not aligned with the Cartesian axes, the absolute permeability is a full tensor in the Cartesian system.

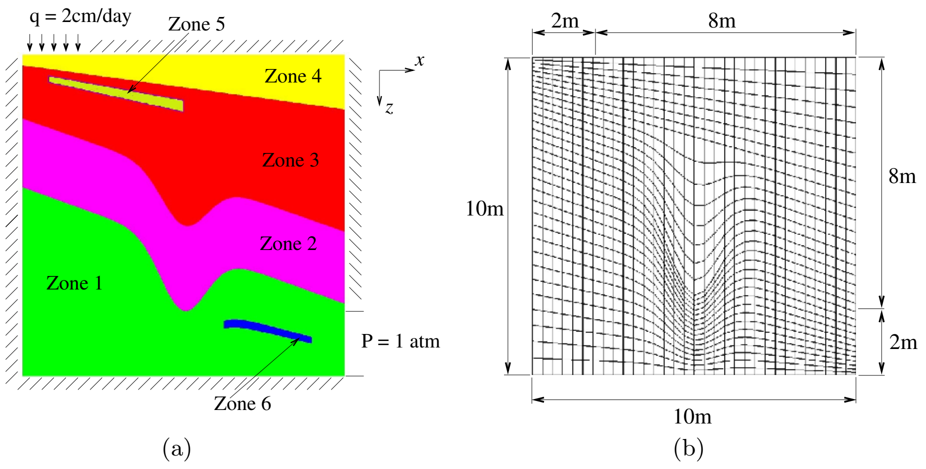


Fig. 2. Computational domain: (a) sloping layered structure with strongly anisotropic inclusions in Zone 5 and 6, (b) a structured mesh 32x32 is mapped onto the stratigraphy. The fine 128x128 computational mesh is constructed in a similar way.

To complete the definition of the two-phase flow model given in Section 2, we define the relative permeability curves as,

$$k_{rw}(S) = (S^*)^2, \qquad k_{ra}(S) = (1 - S^*)^2, \qquad S^* = \frac{S - S_{wr}}{1 - S_{wr} - S_{nwr}}, \tag{7}$$

where S^* is the normalized wetting phase saturation, and $S_{wr} = S_{nwr} = 0.1$ are the residual saturations of the wetting and non-wetting phases, respectively. In

Table 1. Absolute hydraulic permeability tensors are diagonal in the local coordinate system of each zone. Note that the anisotropy ratio is 100 within the inclusions (Zones 5 and 6) and 50 in Zone 2.

Zone	$\mathbb{K}_x(\text{m}^2)$	$\mathbb{K}_z(\text{m}^2)$	$\mathbb{K}_x/\mathbb{K}_z$
1	4×10^{-12}	4×10^{-12}	1
2	5×10^{-12}	1×10^{-13}	50
3	5×10^{-12}	5×10^{-12}	1
4	9×10^{-12}	9×10^{-12}	1
5	5×10^{-11}	5×10^{-13}	100
6	5×10^{-11}	5×10^{-13}	100

addition, the initial saturation is set to a constant, $S(t = 0) = S_{wr} = 0.1$; the porosity is constant, $\phi = 0.2$; and the phase viscosities are $\mu_w = 1.0 \times 10^{-3}$, and $\mu_a = 1.8 \times 10^{-5}$, both in units of $\text{kg} \cdot (\text{m} \cdot \text{s})^{-1}$. Boundary conditions are shown schematically in Fig.2(a) with no-flow everywhere except for the water source in the upper left and the fixed pressure in the bottom right. Fig. 2(b) shows the dimensions of the domain (10m×10m), with the infiltration and fixed pressure conditions spanning 2m each.

To study the efficiency and accuracy of the M^3 method for this infiltration problem, we generate a reference solution using the standard IMPES method on a 128×128 mesh. The saturation at T=6 days is plotted in Fig. 3, where lighter colors represent higher water saturation. Here the rapid flow along both highly anisotropic inclusions is apparent. Also evident is the influence of gravity, which has quickly drawn the fluid down once it passed through the first inclusion. In fact, this *arm* is now reconnecting with the main flow.

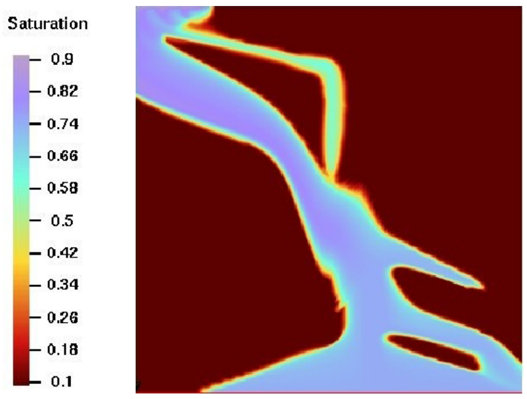


Fig. 3. Reference solution: water saturation profile at $T = 6$ days computed on mesh 128x128 with the fine-scale IMPES method

Since the fine-mesh is a logically structured quadrilateral mesh, the simple aggregation strategy shown in Fig. 1 is readily applied. Thus, we coarsen by a factor of two in each direction, recursively, until the desired total coarsening factor has been reached. We set the mobility threshold, $\epsilon_\lambda = 0.1$, which controls adaptive local updates of the hierarchy. Global updates of the flux coarsening factors (i.e., the α's) are made uniformly in time. The tolerance of the PCG(AMG) iteration for these updates is $\epsilon_r = 10^{-3}$. The time step is adapted according to the stability analysis developed by Coats [16].

We conduct two series of tests with this configuration of the M^3 method. First we consider a fixed total coarsening factor of 32 in each coordinate direction, which gives 4×4 coarse-grid for the pressure solve. We integrate to a time of T=6 days, and examine the affect of increasing the number of updates to the α's, from 12, to 60, and finally 120. The saturation at T=6 for these three scenarios are shown in Fig4(a)-(c). Comparing the result with only 12 updates in Fig4(a), to the reference solution in Fig.3, we see that a number of features have evolved either too slowly (e.g., the flow along the upper inclusion), or too quickly (e.g., the closure of the unsaturated space under the lower inclusion). Increasing to 60 updates, Fig.4(b) begins to correct these problems, but there are still significant differences in the flow along the upper inclusion. Finally, Fig.4(c) shows that with 120 updates we recover accuracy comparable to the fine-scale solution, accurately capturing the highly anisotropic flow through both the upper and lower inclusions. To understand the cost of these multiscale simulations relative to the fine-scale IMPES simulation, it is useful to note that these update scenarios correspond to an average of 2972.7, 613.6 and 320.7 time-steps per update, respectively. Moreover, these updates are based on approximate solutions computed using only 5-10 V-cycles each. The large total coarsening factor leads to an overall speedup of more than 5 times, for all update scenarios considered here. Overall, the computational cost of the pressure solver is now negligible, compared to the cost of the transport.

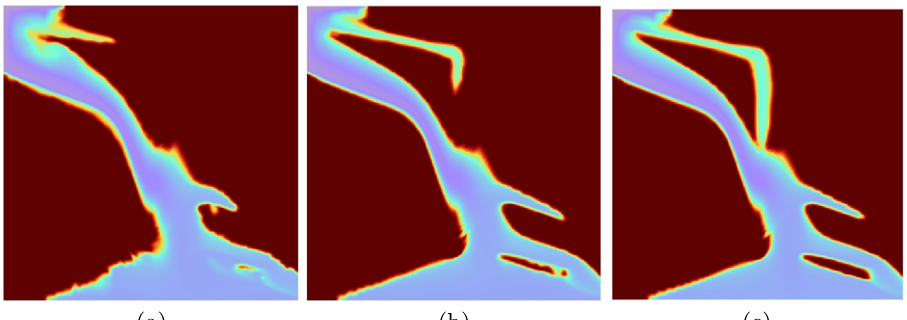


Fig. 4. Multiscale solutions for different numbers of updates: (a) coarsest mesh 4x4, 2972.7 time-steps per update, (b) coarsest mesh 4x4, 613.6 time-steps per update, (c) coarsest mesh 4x4, 320.7 time-steps per update

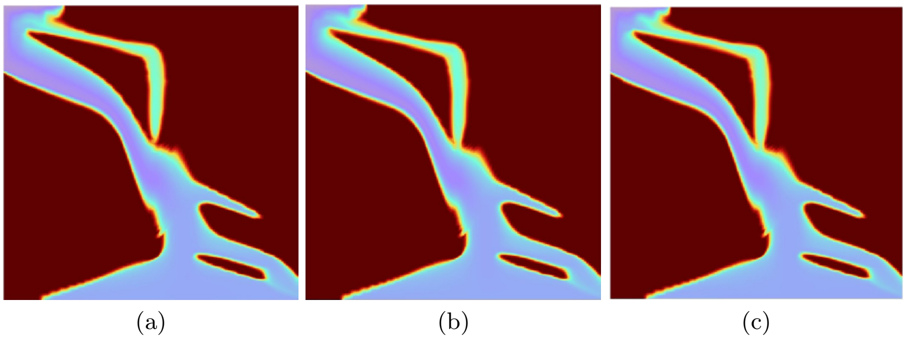


Fig. 5. Multiscale solutions for different coarsening factors: (a) coarsest mesh 16x16, 275.5 time-steps per update, (b) coarsest mesh 8x8, 270.7 time-steps per update, (c) coarsest mesh 4x4, 265.8 time-steps per update

In the second set of tests we integrate to T=6 days with the number of updates of the α's fixed at 150, and consider different total coarsening factors of 8, 16, and 32. The results are shown in Fig.5(a)-(c) and confirm that accuracy comparable to the fine-scale solution has been achieved in all cases. This demonstrates that with an appropriate number of updates the accuracy of multiscale solution deteriorates very slowly as the total coarsening factor increases. The M^3 method is very flexible and robust with respect to the total coarsening factor, and hence, it may be applied to very large-scale simulations where other two-level upscaling methods are insufficient.

5 Conclusion

In this research we extended the M^3 method to include gravity, and studied its effectiveness on a challenging anisotropic infiltration problem. We created a porous medium with sloping non-uniform layers and two strongly anisotropic inclusions, and we used a logically structured mapped grid to capture these features. The M^3 method performed very well, achieving accuracy comparable to the fine-scale solution, at a fraction of the computational cost. As in the original study [1], updating the flux coarsening parameters enabled the use of large coarsening factors, a factor of 32 here, while maintaining both accuracy and overall efficiency. This further highlights the importance of the global information that is provided by the updates of the flux coarsening parameters (the α's). In the future we plan to develop an error indicator that will guide a time-adaptive update strategy for these parameters. We anticipate that this advance will further improve our ability to balance accuracy and efficiency. In addition we are interested in extending this hierarchical methodology to upscale the transport equation.

Acknowledgements. Funded by the Department of Energy at Los Alamos National Laboratory under contracts DE-AC52-06NA25396 and the DOE Office of Science Advanced Computing Research (ASCR) program in Applied Mathematical Sciences. The authors thank George Zyvoloski (LANL) for suggestions and fruitful discussions relating to the numerical experiment.

References

1. Svyatskiy, D., Lipnikov, K., Moulton, J.D.: A Multilevel Multiscale Mimetic (M^3) method for two-phase flows in porous media. J. Comput. Phys. 227, 6727–6753 (2008)
2. Hou, T.Y., Wu, X.H.: A multiscale finite element method for elliptic problems in composite materials and porous media. J. Comput. Phys. 134, 169–189 (1997)
3. Aarnes, J.E., Kippe, V., Lie, K.A.: Mixed multiscale finite elements and streamline methods for reservoir simulation of large geomodels. Adv. Water Resour. 28, 257–271 (2005)
4. Hughes, T.J.: Multiscale phenomena: Green's functions, the dirichlet-to-neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. Computer Methods in Applied Mechanics and Engineering 127, 387–401 (1995)
5. Arbogast, T., Pencheva, G., Wheeler, M.F., Yotov, I.: A multiscale mortar mixed finite element method. Multiscale Model. Simul. 6, 319–346 (2007)
6. Jenny, P., Lee, S.H., Tchelepi, H.A.: Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. J. Comput. Phys. 187, 47–67 (2003)
7. Kuznetsov, Y.A.: Mixed finite element method for diffusion equations on polygonal meshes with mixed cells. J. Numer. Math. 14, 305–315 (2006)
8. Lunati, I., Jenny, P.: Treating highly anisotropic subsurface flow with the multiscale finite-volume method. Multiscale Model. Sim. 6, 308–318 (2007)
9. Forsyth, P.A., Wu, Y.S., Pruess, K.: Robust numerical methods for saturated-unsaturated flow with dry initial conditions in heterogeneous media. Adv. Water Resour. 18, 25–38 (1995)
10. Lunati, I., Jenny, P.: Multiscale finite-volume method for density-driven flow in porous media. Computat. Geosci. 12, 337–350 (2008)
11. Forsyth, P.: A control volume finite element approach to NAPL groundwater contamination. SIAM J. Sci. Stat. Comput. 12, 1029–1057 (1991)
12. Brezzi, F., Lipnikov, K., Simoncini, V.: A family of mimetic finite difference methods on polygonal and polyhedral meshes. Math. Mod. Meth. Appl. S. 15, 1533–1551 (2005)
13. Jenny, P., Lee, S.H., Tchelepi, H.A.: Adaptive multiscale finite-volume method for multiphase flow and transport in porous media. Multiscale Model. Sim. 3, 50–64 (2004)
14. Ruge, J.W., Stüben, K.: Algebraic multigrid (AMG). In: McCormick, S.F. (ed.) Multigrid Methods. Frontiers in Applied Mathematics, vol. 3, pp. 73–130. SIAM, Philadelphia (1987)
15. Zyvoloski, G.A., Vesselinov, V.V.: An investigation of numerical grid effects in parameter estimation. Ground Water 44, 814–825 (2006)
16. Coats, K.H.: Impes stability: Selection of stable timesteps. SPE J 8, 181–187 (2003)

Computational Upscaling of Inertia Effects from Porescale to Mesoscale

Małgorzata Peszyńska[1], Anna Trykozko[2], and Kyle Augustson[3]

[1] Oregon State University, Department of Mathematics,
mpesz@math.oregonstate.edu
[2] Interdisciplinary Centre for Math. and Comp. Modeling, University of Warsaw
aniat@icm.edu.pl
[3] University of Colorado at Boulder
kyle.augustson@Colorado.edu

Abstract. We propose algorithms for computational upscaling of flow from porescale (microscale) to lab scale (mesoscale). In particular, we solve Navier-Stokes equations in complex pore geometries and average their solutions to derive properties of flow relevant at lab scale such as permeability and inertia coefficients. We discuss two variants of traditional discretizations: a simple algorithm which works well in periodic isotropic media and can be used when coarse approximations are needed, and a more complex one which is well suited for nonisotropic geometries. Convergence of solutions and averaging techniques are major concerns but these can be relaxed if only mesoscopic parameters are needed. The project is a proof-of-concept computational laboratory for porous media which delivers data needed for mesoscale simulations by performing microscale computational simulations.

1 Introduction

Computational modeling of flow in porous media such as aquifers and oil-gas reservoirs has been constrained until recently to the scales of physical observation and of experiments such as Darcy-scale (lab-scale $\equiv$ mesoscale). For simulations in large porous reservoirs it has been necessary to *upscale* the models and parameters of flow to macroscale.

In this paper we pursue the upscaling from microscale, i.e., porescale to mesoscale (lab or Darcy scale). While relevant mathematical theory was developed decades ago via homogenization [1] and volume–averaging [2], the computational modeling at porescale had remained unfeasible until recently when advances in micro-imaging were accompanied by increases in computational power and development of discrete models such as network and lattice models [3,4,5,6].

In this paper we are interested in continuum models, i.e., traditional discretizations of partial differential equations adapted to porescale such as studies in [7,8]. We investigate conditions under which simple algorithms can be used efficiently to deliver reliable quantitative information from microscale to mesoscale. Our project can be seen as a first step of a computational laboratory for modeling flow over a range of scales; the model at mesoscale can be further upscaled to macroscale following e.g. [9,10].

The following problem is of interest for stationary incompressible viscous flow. It is well known that linear flow models are not valid beyond certain Reynolds numbers [11,12,13,14,15,16]. At porescale the relevant linear and nonlinear models are the Stokes and Navier-Stokes equations, respectively. At mesoscale those are Darcy and non-Darcy models where it is now believed that in the nonlinear laminar regime it is the inertia rather than micro-turbulence effects that are most important. However, the identification of a particular transition regime between Darcy and non-Darcy flow model as well as a universal mathematical form of an anisotropic non-Darcy model are still a subject of current research and controversies [2,16,17,18,19,20,21], and the values of associated coefficients reported in literature vary significantly.

Consider then a scenario in which data for linear flow at Darcy scale, i.e. permeability $\mathbf{K}$, is known, but no data for modeling inertia effects (denoted by β) is available. We propose to i) use a computational porescale model with inertia effects from which we ii) derive data β for nonlinear models at Darcy scale. In i)-ii) we focus on 2D porescale models with isotropic mesoscale but iii) one can extend i)-ii) to anisotropic nonlinear laws at mesoscale that emerge from complicated anisotropic porescale geometries, and thereby aid current theoretical developments.

The main difficulties of i)-iii) are the following. First, standard discretization techniques for Navier-Stokes equations are well studied but their use in complex geometries requires fine grids and is in general nontrivial. Second, calculating average quantities from computational data is only superficially straightforward since the stability of results with respect to grids and algorithms over a large range of Reynolds numbers must be ensured. Next, as concerns iii), realistic data on porescale geometries such as [5] should be used, and their uncertainty, and dependence of results on averaging regions needs to be accounted for. Finally, computational efficiency of the proposed "on-demand" porescale modeling laboratory must be considered.

In this paper we focus on a proof-of-concept realization of i)-ii); details on iii) will be addressed in a forthcoming paper. In Sections 2 and 3 we describe the relevant physical and computational models, respectively. In Section 4 we propose the method of upscaling and in Section 5 we discuss the results.

2 Computational Models

Let $\Omega \subset \mathbb{R}^d, d = 2, 3$, be an open bounded domain occupied by porous medium and the fluid within. Let $\Omega_F \subsetneq \Omega$ be the part of Ω occupied by the fluid, that is, the domain of flow, and let rock (solid) part be $\Omega_R = \Omega \setminus \Omega_F$. Let $\partial\Omega$ denote the boundary of Ω, and let $\Gamma = \partial\Omega_F \setminus \partial\Omega$ be the interior boundary (between rock and fluid domains) while the external boundary of flow $\partial\Omega_F \cap \partial\Omega$ is divided into inflow Γ_{in} and outflow Γ_{out} parts. We also denote by η the unit outward normal to the boundary and by τ the unit tangent. For simplicity no special notation is used for numerical solutions.

Flow at porescale. We consider an incompressible Newtonian fluid of velocity $\mathbf{u}$ and pressure p flowing in Ω_F. The fluid's viscosity is denoted by μ and the density $\rho \equiv 1$. We consider flow driven primarily by external boundary conditions, such as in a lab core. We prescribe velocity at the inlet and impose an outflow condition at the outlet. On internal boundaries we assume no-slip condition $\mathbf{u} = 0$. We assume that the Reynolds number $\mathbf{Re}$ is correlated to the magnitude of inflow velocities.

At microscale (porescale), for steady-state flow, in the absence of forces and mass source/sink terms, the momentum and mass conservation in Eulerian frame are expressed by Navier-Stokes equations and continuity equation [14]

$$\mathbf{u} \cdot \nabla \mathbf{u} - \mu \triangle \mathbf{u} = -\nabla p, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

In 2D ($d = 2$) it is convenient to consider the formulation in terms of the vorticity vector $\omega = \nabla \times \mathbf{u}$ and the (scalar) stream function ψ defined by $\mathbf{u} = \nabla \times \psi$ [14]. Taking $\nabla\times$ equation (1) and noticing $\nabla \times (\nabla p) = 0$, one obtains the system

$$\boldsymbol{u} \cdot \nabla \boldsymbol{\omega} = \mu \Delta \boldsymbol{\omega}, \tag{3}$$

$$\Delta \psi = -\omega. \tag{4}$$

The last equation follows from standard calculation $\boldsymbol{\omega} = \nabla \times (\nabla \times \boldsymbol{\psi}) = \nabla(\nabla \cdot \boldsymbol{\psi}) - \Delta \boldsymbol{\psi}$, which, with (2), for 2D flow reduces to (4). We can get p from

$$-\Delta p = (\nabla(\boldsymbol{u} \cdot \nabla)) \cdot \boldsymbol{u} = 2\left(\frac{\partial u_x}{\partial x}\frac{\partial u_y}{\partial y} - \frac{\partial u_x}{\partial y}\frac{\partial u_y}{\partial x}\right) \tag{5}$$

For small $\mathbf{Re}$ the nonlinear convective terms associated with $\mathbf{u}\cdot$ are dropped from (1) and (3) and we have the (linear) Stokes approximation

$$-\mu \triangle \mathbf{u} = -\nabla p, \tag{6}$$

which is valid when viscous effects dominate in the flow. For larger $\mathbf{Re}$ the inertia effects associated with $\mathbf{u}\cdot$ cannot be neglected. We recall that, up to the definition/units of characteristic quantities, the linear laminar flow regime is for $\mathbf{Re} < 1$, the nonlinear regime is for $1 \leq \mathbf{Re} < 100$, and that turbulence may occur for $\mathbf{Re} \geq 100$ [13,16]; however, turbulence rarely occurs in porous media.

Flow at Darcy scale. At mesoscale the boundaries between Ω_F and Ω_R are no more recognized, One considers an average pressure $P :=< p >$ and velocity (flux) $\mathbf{U} :=< \mathbf{u} >$ where the averages over a volume $V(\mathbf{x})$ centered at $\mathbf{x} \in \Omega$ are defined as $< q >_V \equiv < q > (x) \equiv \frac{1}{|V(x)|} \int_{V(x)} q(y)dy$. (In what follows we drop subscript V). Conservation of mass after averaging yields $\nabla \cdot < \mathbf{u} >= 0$; note that derivatives in ∇ are taken with respect to large scale variable $\mathbf{x}$. The flow in Ω at mesoscale is driven by boundary conditions which are averages of porescale external boundary conditions.

Darcy's law is a linear momentum equation at mesoscale which can be proven [22,1] to be an average of Stokes flow (6)

$$\mu \mathbf{K}^{-1} < \mathbf{u} >= -\nabla < p > . \tag{7}$$

where the values of a symmetric permeability tensor $\mathbf{K}$ are measured in a lab and have been obtained experimentally for many porous materials [16]. $\mathbf{K}$ reflects geometry at porescale, and $\mathbf{K}^{-1}$ measures resistance of porous medium to the flow. For heterogenous media $\mathbf{K} = \mathbf{K}(\mathbf{x})$, for isotropic media $\mathbf{K} = K\mathbf{I}$. Due to large viscous dissipation and interstitial effects common in porous media, Darcy's law is a good approximation for a large class of flow phenomena.

In the nonlinear laminar regime with significant inertia, averaging (1) yields

$$\mathcal{K}^{-1}(< \mathbf{u} >) < \mathbf{u} >= -\nabla < p > . \tag{8}$$

The 1D model for $\mathcal{K}$ first proposed by Forchheimer [11] was $\mathcal{K}^{-1}(U) := \mu K^{-1} + \beta|U|$, while multidimensional isotropic version [12,16,13,15] reads

$$\mathcal{K}^{-1}(< \mathbf{u} >) < \mathbf{u} >:= \left(\mathbf{K}^{-1}\mu + \beta| < \mathbf{u} > |\right) < \mathbf{u} >= - < \nabla p > . \tag{9}$$

The form of nonlinear map $\mathcal{K}$ for general anisotropic 2D and 3D media has been the subject of theoretical research for general anisotropic 2D and 3D media [17,8,21,18,20,19]. Even if the model (9) for $\mathcal{K}$ is accepted, the data for coefficient β are not universally available and/or consistent.

Mathematical upscaling from micro- to mesoscale. There are essentially two methodologies that apply. The first, with the use of homogenization theory (H) [22,1], requires periodic geometry but gives elegant theorems on convergence of the averages of microscale quantities to the appropriate mesoscale quantities when the size of periodic cell goes to 0. The second, volume averaging (VA), does not restrict geometry and proposes that the averaged quantities are reasonably stable if the averaging region (REV ≡ Representative Elementary Volume) is large enough [19]. However, it may be difficult to quantify what size of REV is sufficient; see [23,24] for related work on elasticity and our forthcoming work.

3 Computational Models for Porescale

For the flow in Ω_F we consider two algorithms $\mathcal{H}$ and $\mathcal{VA}$ described below which are useful in similar contexts as, respectively, the mathematical upscaling methods H and VA. For other algorithms see [25,26].

We illustrate the algorithms with the following scenario. All flow in Ω is from left to right. The pore geometries are idealized: we envision rock grains as very long cylinders so that every cross-section can be approximated by a 2D computational region with Ω_R being a union of solid disks replicated periodically. The ratio of disk diameters to the size of the period denoted by D ranges from 0.6 to 0.9 in this study; see Fig. 1 and 3.

Algorithm $\mathcal{H}$. This simple algorithm solves for (ω, ψ) in $d = 2$ and uses simple structured grids over Ω_F and therefore can be easily adapted to interpret data from porescale imaging [5] without significant investment of time in grid generation. It is based on a central finite difference formulation enhanced by treatment of boundary conditions and post-processing, following [27,28,29].

The discretization of (3) and (4) yields

$$(\mu\Delta_h - (\boldsymbol{u} \cdot \nabla_h))\,\omega = 0, \tag{10}$$

$$\Delta_h\psi = -\omega. \tag{11}$$

where the numerical Laplacian Δ_h has the usual 5-point stencil and the advective term is computed using second order central differences. The coupling in the model is resolved by iteration: given ω^n, i) compute ψ^{n+1} from (11), then ii) calculate velocity $\boldsymbol{u}^{n+1}$ from ψ^{n+1}, then iii) solve (10) for $\omega^{n+\frac{1}{2}}$, and finally iv) compute $\omega^{n+1} = \lambda\omega^{n+\frac{1}{2}} + (1-\lambda)\omega^n$ where λ is the relaxation parameter.

In this algorithm steps i) and ii) require that boundary values of ψ and ω are known, respectively. This is the most delicate part of the algorithm and crucial for porescale computations: an idea how to implement inlet, outlet, and no-slip conditions follows; we refer to [26,28,29] for details.

For structured rectangular grids over a periodic cell Ω, the boundary Γ is composed of vertical and horizontal segments only, and the external boundary has either vertical or horizontal $\Gamma_{in}, \Gamma_{out}$. Consider vertical Γ_{in}. On inlet boundary, $\mathbf{u} = (u_x, 0)$ is given and hence ψ has to be constant and given as the integral of u_x while vorticity is given from (4). On the vertical portion of the outlet boundary we have $\frac{\partial\omega}{\partial x} = 0$ and $\frac{\partial\psi}{\partial x} = 0$.

In the interior, we have no-slip boundary conditions $\mathbf{u} = 0$; it follows that $\psi \equiv const.$ To find useful conditions for ω, we approximate its second derivative as follows. Consider Taylor expansion $\psi(x+\delta x, y) = \psi(x,y) + \delta x\frac{\partial\psi}{\partial x} + \frac{\delta x^2}{2}\frac{\partial^2\psi}{\partial x^2} + O(\delta x^3)$. But $u_y = -\frac{\partial\psi}{\partial x} = 0$ thus $-\omega = \frac{\partial^2\psi}{\partial x^2} = \frac{2(\psi(x+\Delta x,y)-\psi(x,y))}{\delta x^2} + O(\delta x)$.

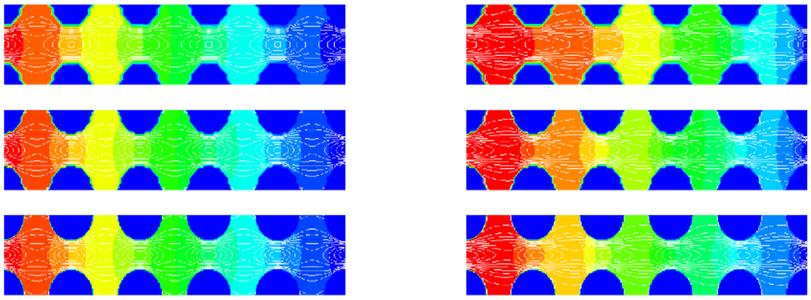


Fig. 1. Results of algorithm $\mathcal{H}$, $D = 0.6$. Shown are profiles of p overlaying contours of $\mathbf{u}$ for $\mathbf{Re} = 1, 100$, (left and right) on three grids: coarse, fine, and very fine (top to bottom). Even though the pointwise values appear unresolved on the coarse meshes, the computed averages and $\mathbf{K}$ and β are stable on all grids.

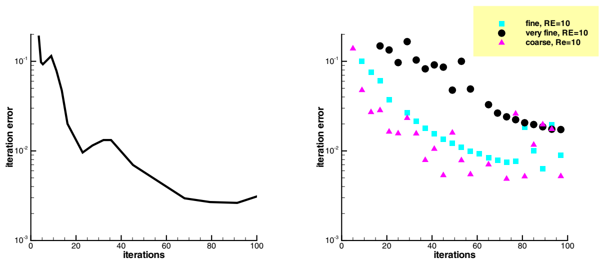


Fig. 2. Convergence of relaxation iteration in algorithm $\mathcal{H}$. Left: standard benchmark problem of Poiseille flow [26]. Right: flow in geometry as in Fig. 1. The iteration error (both) is defined as discrete l_2 norm of the stream function ψ.

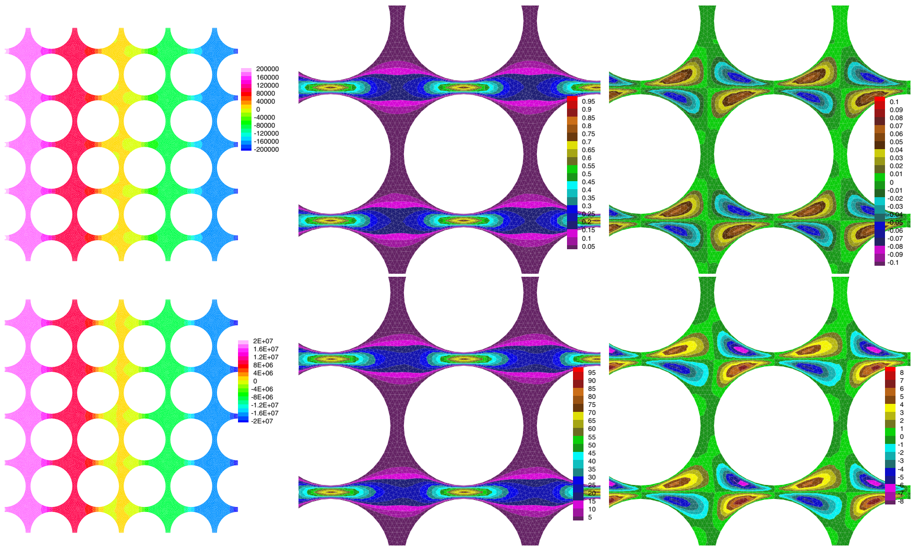


Fig. 3. Results of $\mathcal{VA}$ for $D = 0.9$ with $\mathbf{Re} = 1$ (top) and $\mathbf{Re} = 100$ (bottom). Shown are contours of pressure p (left) and zoomed in velocity $\mathbf{u}$ profiles: component x (middle) and y (right).

To recover (post-process for) the velocities and pressures, we use central finite differences in the interior of the domain and appropriate one-sided differences at the boundary. The pressure is found from (5).

As seen in Fig. 1, the algorithm works reasonably well for a range of Reynolds numbers and grids that are not very fine. The difficulties arise since finding an optimal value of λ in step iv) may be a problem; see Fig. 2. The simplicity of $\mathcal{H}$ is in that it consequently uses the same Poisson solver for which very efficient solvers and preconditioners [26,25] are available. As an alternative, a coupled solver for (10)–(11) can be written but this requires sophisticated nonsymmetric solvers and preconditioners. Overall, the algorithm $\mathcal{H}$ works well for small periodic domains Ω but may scale poorly to large regions, complex geometries, and large **Re**.

Algorithm $\mathcal{VA}$. This algorithm solves for $(\mathbf{u}, p)$ and can be used in complicated geometries but requires substantial pre-processing; it follows an industry standard in computational fluid dynamics [30,31]; general unstructured grids can be used in 2D and 3D. We omit the details but provide an example which illustrates the grids and complexity of computations, see Fig. 3.

4 Upscaling Algorithms from Porescale to Mesoscale

Strictly speaking, the work reported in this paper does not require any computations at mesoscale, i.e., in Ω. However, keeping in mind our future goals, we choose to upscale from microscale to *some chosen* computational grid at mesoscale. In this paper we choose the conservative cell-centered finite difference method equivalent to lowest order Mixed Finite Element method on rectangles [32]; this provides a bridge to macroscale following [9,10].

The idea is as follows: we impose a mesoscale cell-centered grid over Ω in a way which defines principal directions of flow that we anticipate will prevail at mesoscale (This may help to avoid handling full tensor $\mathbf{K}$ at mesoscale). With each center of mesocale grid (X_j, Y_j), we have an associated cell Ω_j over which we average to get values P_j. Velocities are computed over unions of regions so that they are associated with locations "dual" to those for pressures [32]; see Fig. 4 for illustration. Ideally, the locations (X_j, Y_j) coincide with centers of mass of Ω_j and the velocity components are computed in the direction of principal axis. However, there are ways to handle situations when this does not hold.

Assume now that porescale results $(\mathbf{u}, p)$ are available. We then compute $\mathbf{U}$ and P as discussed above. Next, by inverse modeling, we identify resistance of the medium $\mathcal{K}^{-1}$ in the discrete counterpart of (8).

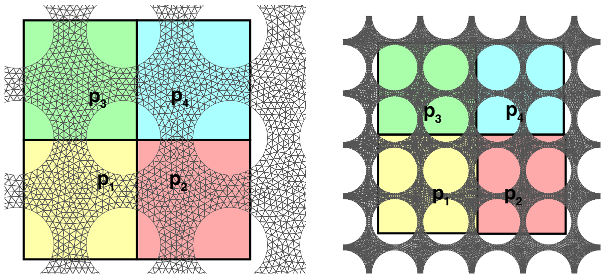


Fig. 4. Left: averaging region for a small periodic region Ω, case $D = 0.7$. The x-component $\mathbf{U}$ can be computed from averaging over regions Ω_1, Ω_3 and Ω_2, Ω_4. Right: general averaging region, case $D = 0.9$.

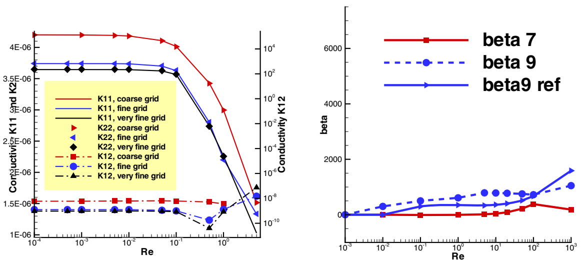


Fig. 5. Results of upscaling from porescale to lab scale. Left: dependence of $\mathbf{K}$ on the grid and $\mathbf{Re}$; notice isotropy at mesoscale revealed by equal diagonal and very small off-diagonal components of $\mathcal{K}$. Right: dependence of β on the relative diameter $D \in (0.7, 0.9)$ and on grid size (results computed for coarse and fine grids).

Note that for small $\mathbf{Re}$, the resistance $\mathcal{K}^{-1}$ reduces to $\mathbf{K}^{-1}$ as in (7). Thus, if data is available for a large range of $\mathbf{Re}$ from creeping flow to nonlinear laminar regime, then one could hope to identify the appropriate model of tensor $\mathcal{K}$. In particular, if the medium is isotropic at mesoscale, then (9) is valid. In other words, given $\mathcal{K}$ and $\mathbf{K}$ and knowing $\mathbf{U}$, one can compute β for any $\mathbf{Re}$. Clearly if the model for $\mathcal{K}$ is valid and the computational algorithm is successful then β remains reasonably constant throughout the nonlinear laminar regime; this appears true in our results, see Fig. 5.

We stress that stability of $\mathbf{K}$ and β is not guaranteed with just any ad-hoc averaging technique; in particular, the choice of REV, principal axis and their orientation, and of the boundary conditions, plays a significant role. In addition, there is currently no general explicit mathematical model and virtually no experimental work for anisotropic inertia represented by $\mathcal{K}$.

5 Discussion

Fluid flow in porespace is subject to viscous effects, inertia effects, and dissipation on the solid boundaries. In order to approximate the flow accurately for large $\mathbf{Re}$, we need to ensure that the grid is fine enough in the channels where the solid boundaries are the closest to each other.

Using algorithm $\mathcal{H}$ we observed reasonable convergence for $D \leq 0.6$. However, more work needs to be done before the algorithm $\mathcal{H}$ can scale to more complicated geometries with $D > 0.6$ and for large $\mathbf{Re}$. In particular, we are considering a transient regularization of (3) which will help the convergence.

The use of algorithm $\mathcal{VA}$ was promising for realistic porosities i.e. $D > 0.9$. However, $\mathcal{VA}$ requires care in gridding and monitoring convergence of the iterations. Here the difficulties are related to proper porescale grid definition with respect to principal axis. There is also the relative lack of availability of $\mathcal{VA}$ due to its commercial implementation.

Overall, regardless of the porescale algorithm chosen, for some grids and some $\mathbf{Re}$, the profiles of $(\mathbf{u}, p)$ may reveal local instabilities. However, this does not necessarily lead to an instability of mesoscale properties at least with the averaging method that we proposed. In fact, $\mathbf{K}$ and β appear stable for a large range of values of $\mathbf{Re}$ as well as appear convergent with respect to the grid size, see Fig. 5. We note that both $\mathbf{K}$ and β are nonlinear quantities of interest in the sense of [24]; see also [33] for recent related work on multiscale modeling.

Current and future work includes convergence analysis as well as serious computational studies aiding the theoretical modeling of tensor $\mathcal{K}$. Our project is a prototype of a computational laboratory which can provide on-demand model data for flow with inertia in porous media.

Acknowledgements. The work of Peszynska and Augustson was partially supported from NSF grant 0511190 and DOE grant 98089. The authors thank anonymous reviewers for additional references and comments that helped to improve the paper.

References

1. Tartar, L.: Incompressible fluid flow in a porous medium–convergence of the homogenization process. In: Nonhomogeneous media and vibration theory. Lecture Notes in Physics, vol. 127, pp. 368–377. Springer, Berlin (1980)
2. Ruth, D., Ma, H.: On the derivation of the Forchheimer equation by means of the averaging theorem. Transp. Porous Media 7(3), 255–264 (1992)
3. Schaap, M., Porter, M., Christensen, B.S.B., Wildenschild, D.: Comparison of pressure-saturation characteristics derived from computed tomography and Lattice–Boltzmann simulations. Water Resour. Res. 43 (2007)
4. Pan, C., Hilpert, M., Miller, C.: Pore-scale modeling of saturated permeabilities in random sphere packings. Physical Review E. 64 (2001)
5. Lindquist, W.B.: Network flow model studies and 3D pore structure. In: Fluid flow and transport in porous media: mathematical and numerical treatment. Contemp. Math, vol. 295, pp. 355–366. Amer. Math. Soc, Providence (2002)
6. Succi, S.: The Lattice Boltzmann equation for fluid dynamics and beyond. In: Numerical Mathematics and Scientific Computation. The Clarendon Press, Oxford University Press, New York (2001)
7. Andrade, J.S., Costa, U.M.S., Almeida, M.P., Makse, H.A., Stanley, H.E.: Inertial effects on fluid flow through disordered porous media. Phys. Rev. Lett. 82(26), 5249–5252 (1999)
8. Fourar, M., Lenormand, R., Karimi-Fard, M., Horne, R.: Inertia effects in high-rate flow through heterogeneous porous media. Transp. Porous Media 60(18), 353–370 (2005)
9. Durlofsky, L.J.: Numerical calculation of equivalent grid block permeability tensors for heterogeneous porous media. Water Resour. Res. 27(5), 699–708 (1991)
10. Garibotti, C., Peszynska, M.: Upscaling Non-Darcy Flow. Transport in Porous Media (published online March 13, 2009) doi:10.1007/s11242–009–9369–2
11. Forchheimer, P.: Wasserbewegung durch Boden. Zeit. Ver. Deut. Ing. (45), 1781–1788 (1901)
12. Ergun, S.: Fluid flow through packed columns. Chemical Engineering Progress 48, 89–94 (1952)

13. Scheidegger, A.E.: The physics of flow through porous media. The Macmillan Co., New York (1960)
14. Batchelor, G.K.: An introduction to fluid dynamics. Cambridge (1999)
15. Dullien, F.: Porous media. Academic Press, San Diego (1979)
16. Bear, J.: Dynamics of Fluids in Porous Media. Dover, New York (1972)
17. Mei, C.C., Auriault, J.L.: The effect of weak inertia on flow through a porous medium. J. Fluid Mech. 222, 647–663 (1991)
18. Giorgi, T.: Derivation of Forchheimer law via matched asymptotic expansions. Transp. Porous Media 29, 191–206 (1997)
19. Bennethum, L.S., Giorgi, T.: Generalized Forchheimer equation for two-phase flow based on hybrid mixture theory. Transp. Porous Media 26(3), 261–275 (1997)
20. Chen, Z., Lyons, S.L., Qin, G.: Derivation of the Forchheimer law via homogenization. Transp. Porous Media 44(2), 325–335 (2001)
21. Huang, H., Ayoub, J.: Applicability of the Forchheimer equation for non-Darcy flow in porous media. SPE Journal (SPE 102715), 112–122 (2008)
22. Bensoussan, A., Lions, J.L., Papanicolaou, G.: Asymptotic analysis for periodic structures. North-Holland Publishing Co., Amsterdam (1978)
23. Moës, N., Oden, J.T., Vemaganti, K., Remacle, J.F.: Simplified methods and a posteriori error estimation for the homogenization of representative volume elements (RVE). Comput. Methods Appl. Mech. Engrg. 176(1-4), 265–278 (1999)
24. Oden, J.T., Vemaganti, K.S.: Estimation of local modeling error and goal-oriented adaptive modeling of heterogeneous materials. I. Error estimates and adaptive algorithms. J. Comput. Phys. 164(1), 22–47 (2000)
25. Wesseling, P.: Principles of computational fluid dynamics. Springer Series in Computational Mathematics, vol. 29. Springer, Berlin (2001)
26. Elman, H.C., Silvester, D.J., Wathen, A.J.: Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Oxford University Press, New York (2005)
27. Landau, R.H., Páez, M.J.: Computational physics. Wiley, New York (1997)
28. Spotz, W.F., Carey, G.F.: High-order compact scheme for the steady streamfunction vorticity equations. Internat. J. Numer. Methods Engrg. 38(20), 3497–3512 (1995)
29. Weinan, E., Liu, J.G.: Vorticity boundary condition and related issues for finite difference schemes. J. Comput. Phys. 124(2), 368–382 (1996)
30. Patankar, S.V.: Numerical Heat Transfer and Fluid Flow FLUENT software, Hemisphere, Washington DC (1980)
31. Fluent Inc.: Fluent User's Guide, ver. 6.3 (2006)
32. Russell, T.F., Wheeler, M.F.: Finite element and finite difference methods for continuous flows in porous media. In: Ewing, R.E. (ed.) The Mathematics of Reservoir Simulation, pp. 35–106. SIAM, Philadelphia (1983)
33. Oden, J.T., Prudhomme, S., Romkes, A., Bauman, P.T.: Multiscale modeling of physical phenomena: adaptive control of models. SIAM J. Sci. Comput. 28(6), 2359–2389 (2006)

Towards a Complex Automata Multiscale Model of In-Stent Restenosis

Alfonso Caiazzo[1], David Evans[2], Jean-Luc Falcone[3], Jan Hegewald[4], Eric Lorenz[5], Bernd Stahl[3], Dinan Wang[6], Jörg Bernsdorf[6], Bastien Chopard[3], Julian Gunn[2], Rod Hose[2], Manfred Krafczyk[4], Patricia Lawford[2], Rod Smallwood[2], Dawn Walker[2], and Alfons G. Hoekstra[5]

[1] INRIA Rocquencourt, France
{alfonso.caiazzo}@inria.fr
[2] The University of Sheffield, UK
{david.evans,j.gunn,d.r.hose,p.lawford,r.smallwood, d.c.walker}@sheffield.ac.uk
[3] CUI Department, University of Geneva, Switzerland
{jean-luc.falcone,bernd.stahl,bastien.chopard}@unige.ch
[4] Technical University of Braunschweig, Germany
{hegewald,kraft}@irmb.tu-bs.de
[5] University of Amsterdam, the Netherlands
{e.lorenz,a.g.hoekstra}@uva.nl
[6] NEC Laboratories Europe, St. Augustin, Germany
{d.wang,j.bernsdorf}@it.neclab.eu

Abstract. In-stent restenosis, the maladaptive response of a blood vessel to injury caused by the deployment of a stent, is a multiscale problem involving a large number of processes. We describe a Complex Automata Model for in-stent restenosis, coupling a bulk flow, drug diffusion, and smooth muscle cell model, all operating on different time scales. Details of the single scale models and of the coupling interfaces are described, together with first simulation results, obtained with a dedicated software environment for Complex Automata simulations. The results show that the model can reproduce growth trends observed in experimental studies.

Keywords: Complex Automata, in-stent restenosis, multiscale modeling.

1 Introduction

A *stenosis* is a narrowing of a blood vessel lumen due to the presence of an atherosclerotic plaque. This can be corrected by balloon angioplasty, after which a *stent* (metal mesh) is deployed to prevent the vessel from collapsing. The injury caused by the stent can lead to a maladaptive biological response of the cellular tissue (mainly smooth muscle cells). The abnormal growth can produce a new stenosis (re-stenosis). The multiscale nature of this process has been discussed in detail previously by Evans et al. [1].

The geometry of the stent employed influences the biological events occurring in the vessel following deployment. Stent length, strut thickness, number, cross-sectional shape and arrangement all influence the hemodynamics and degree of injury/stretch observed within the stented segment [2]. These in turn, are critical determinants of the severity of restenosis observed. Additionally, stents may be coated with active compounds targeted at the biological processes responsible for driving the progression of a restenosis which, when eluted locally at the stented site, can prevent proliferation of smooth muscle cells and neointimal growth.

The development of a multiscale *in silico* model capable of testing both the influence of stent geometry and that of drug elution is motivated by the desire for a better understanding of the dynamics regulating restenosis. Thus providing a potentially powerful tool for improved understanding of the biology, and to assist in the process of device/therapy development.

As in many other biological systems, the dynamics of in-stent restenosis span many orders of magnitude through the scales, from the smallest microscopic scales up to the largest macroscopic ones. The wealth of experimental data that is now available has made *in silico* experimentation an attractive tool in systems biology, allowing hypothesis testing and formulation of predictions which can be further tested *in vitro* or *in vivo* [3]. The next challenge is to study, not only fundamental processes, on all these separate scales, but also their mutual coupling across the scales and to determine the emergent structure and function of the resulting system [4].

Based on the conceptual description of the relevant processes and their characteristic scales as presented in [1], we propose a simplified CxA model of in-stent restenosis, coupling a *lattice Boltzmann* bulk flow (BF) solver (for the blood flow), an *agent based model* for smooth muscle cell (SMC) dynamics (simulating growth, cell cycle, physical and biological cell-cell interaction), and a *Finite Difference scheme* for the drug diffusion (DD) within the cellular tissue.

In section 2 we introduce the main ideas behind the CxA approach. In section 3 following a short introduction on instent restenosis, we present the multiscale model. We describe the main characteristics of the single scale solvers, which have been developed independently from one another, and independently from the ultimate application. Then, in section 4 we discuss how the coupling between these models has been realized using a CxA dedicated software environment. Preliminary numerical results are presented in section 5, and conclusions are discussed in section 6.

2 Complex Automata Modeling and Simulation

Recently we introduced Complex Automata (CxA) as a paradigm for multiscale modelling and simulation [5,7,8]. A multiscale system is decomposed into mutually interacting single scale models. The multiscale system can be represented graphically on a Scale Separation Map (SSM), where the horizontal and vertical axes represent the temporal and spatial scales. An example of such SSM is shown in Fig. 1 for the multiscale model of in-stent restenosis (as discussed in detail in section 3). The single scale models and their interactions are drawn on the map. The single scale models are discrete and explicitly update their state to a next time step using a well defined evolution operator, in the form of *collision+propagation*[1]. An essential step in the modelling process is the inclusion of specific *coupling templates*, designed to mimic the dynamic behavior of the multi-scale process as accurately as possible.

The conceptual ideas behind the CxA approach have been used to develop the COAST *Multiscale Coupling Library and Environment* (MUSCLE) [11,12], a software environment where a CxA can be implemented naturally. The coupling library is based on JADE, a multiagent platform, where both the *kernels* (i.e. the single scale models) and the *conduits* (i.e. the multiscale couplings) are treated as agents. The single scale models do not need to be aware of each other, and the information concerning the coupling and the global setup are held by the framework. The structure of the coupling library allows a complete independence from native codes, which can be replaced with a different source, provided the interface with respect to the framework (i.e. the wrapper agent) remains the same. Kernels communicate via *smart conduits* (special agents which implement the multiscale coupling) using two communication primitives, a non-blocking send operation and a blocking receive operation. Often, conduits perform *filtering* operations, in order to match the input and output of the different single scale models. The details of coupling for this particular application are discussed in section 4.

3 Multiscale Model of In-Stent Restenosis

In a previous work [1], the processes key to the restenosis system and their mutual coupling were identified and considered in terms of their temporal and spatial scales, allowing us to generate a comprehensive Scale Separation Map [1]. In this paper, we consider a simplified model focused on SMC behavior, and the interaction with blood flow and drug eluted from the stent. A simplified SSM is shown in Fig. 1. After deployment of the stent, which is actually modelled in a separate box as an initial condition (using the SMC model itself, see section 4), SMCs start to proliferate. This proliferation depends on the blood flow (specifically the wall shear stress) and the concentration of drug (in the case of a drug eluting stent). The blood flow, in turn, depends on the lumen domain (and therefore changes due to the proliferation of SMC), and the concentration of drug depends on the SMC domain itself (and therefore also on SMC proliferation).

In the current model we assume that scale separation between the single scale models is confined to the temporal scale. The SMC proliferation is the slowest process, dictated by the cell cycle whereas flow is a fast process, dictated by one heart beat. Due to the specific value of the diffusion coefficients and the typical size of the tissue level, the temporal scale of the diffusion process resides between

[1] The terminology collision-propagation is borrowed from the lattice gas automata framework (see e.g. [9]), and it has been recently shown to be equivalent to other update paradigms [10].

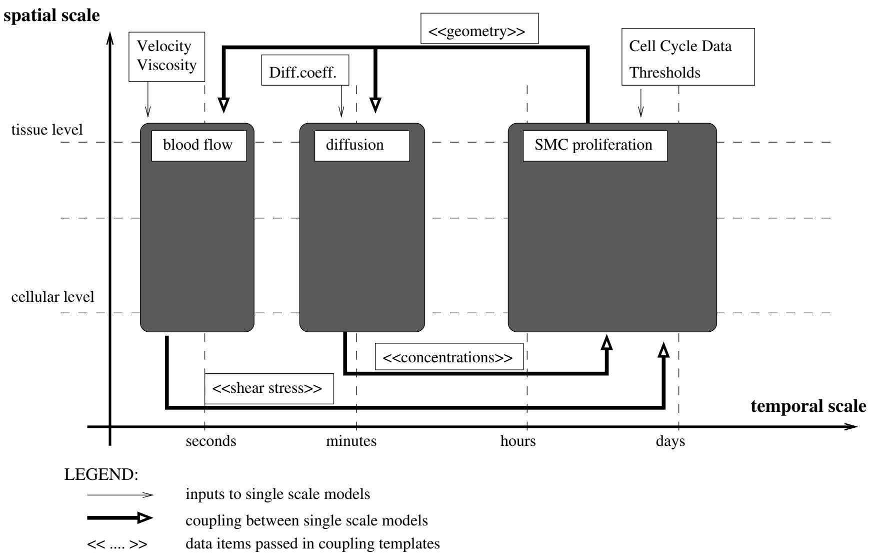


Fig. 1. The simplified SSM

the flow and the SMC scales. In future models we will also explicitly consider spatial scale separation.

The blood flow inside the vessel is modeled as a Newtonian incompressible flow. We employ a lattice Boltzmann Method to obtain a numerical solution (see for example [9,13]). The observable related to the bulk flow (BF) model is the wall shear stress on the vessel boundary (WSS). This is needed as input for the SMC model, after being properly mapped (section 4).

The dynamics of smooth muscle cells are simulated using an Agent Based Model (SMC model), where each cell represents a single agent, identified by a set of state-variables: *position, radius, biological state, drug concentration and structural stress.* Each SMC agent evolves in time according to the current state, and to the states of neighboring cells. The solver comprises a physical solver, simulating the structural dynamics of cells, and a biological solver which simulates the cell cycle, according to a biological rule set. More detail of the SMC model will be published elsewhere [14]. Cells are represented by their centres and potential functions, which determine non-linear repulsive and attractive inter-cell forces. In additional, boundary forces, viscous friction and radial elastic forces modeling the primary fibre direction of real 3D SMCs and motility forces modeling cell migration are taken into account [14]. At every iteration step, new equilibrium positions of SMCs are computed by iterating a finite difference scheme until steady state is reached. Next, the structural stress is calculated and provided as input to the biological solver.

The biological solver contains a model of the cell cycle, consisting of a discrete set of states, a quiescent state **G0**, a growth state **G1** and finally a mitotic state **S/G2/M**, when a *mother* cell eventually divides into two new *daughter* cells. Progression through the cell cycle takes place at a fixed rate, depending on the time step, and culminates in mitosis (division). Cells may enter or leave the inactive phase of the cell cycle (G0) depending on certain rules, which take into account contact inhibition (based on neighbor count), structural stress, and local drug concentration (for all cells) and wall shear stress (WSS) and oscillatory stress index (OSI) for cells in contact with the fluid. Low WSS or high structural strain are individually capable of inducing agent proliferation if drug concentration and contact inhibition criteria allow.

Drug eluting stents represent an effective way of inhibiting neointimal formation. We include this additional aspect in the CxA model by implementing a drug diffusion sub-process. Since drug is eluted from the stent and diffuses into the cellular tissue, the spatial domain for the Drug Diffusion Model coincides with the space occupied by SMCs. The stent struts act as a source, whilst the boundaries between flow and cells are considered sinks (assuming that drug eluted into the lumen is continuously flushed away by the faster blood flow). Since biological tissues are heterogeneous in nature, we assume that this process can be described using a generic anisotropic diffusion.

The diffusion tensor is chosen such that the diffusion along the artery axis or tangentially to a cross section is at least 10 times higher than the diffusion in the radial direction [15,16]. To solve the diffusion equation numerically we use a Finite Differences (FD) approach, solved by a Propagation-Collision loop, which complies with the CxA modeling language. According to [16], the time scale to reach the steady state is of the order of minutes. Therefore, when coupling DD and SMC, we are largely interested in the steady drug concentration (the time step for the SMC model, which uses as input the drug concentration, is of the order of 1 hour).

4 CxA Implementation: Connection Scheme, Kernels, Conduits

In order to combine the single scale models in a CxA setup, we need to define a communication graph, the *Connection Scheme* (CS), which specifies in detail the communication topology of the CxA, i.e. which pairs of single scale models, the *kernels* of the CxA, communicate. For the simplified model of in-stent restenosis, the CS is shown in Figure 2.

At each SMC iteration, the cell configuration defines a new domain for the BF solver. The conduit from SMC to BF converts the array of positions and radii of the cells into a computational mesh, decomposed into fluid and solid nodes for the flow solver.

Similarly, a conduit from SMC to DD converts the array of positions and radii of the cells, into a computational mesh for the drug diffusion solver, marking the nodes as tissue, source, or sink.

plus .1em In some cases, the interaction between kernels is slightly more complex, and multiple inputs are required for computing an output. This is the case for the

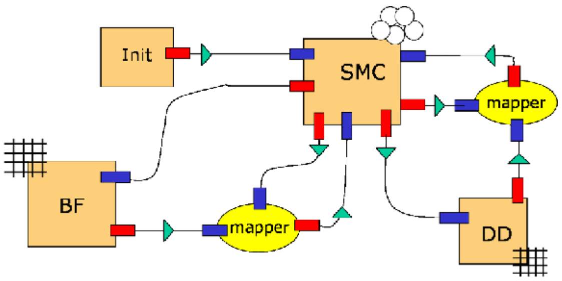


Fig. 2. The Connection Scheme, showing the single scale models (Bulk Flow, SMC, Drug Diffusion), the Init agent (used to generate the initial structural stress condition in the tissue), the mapper agents and the conduits. For each single scale models it is specified whether it is mesh-based (BF, DD) or Agent-based (SMC).

mapper agents in (Fig. 2). For example, since the SMC agents are moving, each time an input for the SMC model has to be computed, the SMC itself has to send its current state, to allow the coupling agent to map the values to each agent.

The values of fluid shear stress at the boundary affect the biological evolution of the cells. Given the fluid output, and the current cell configuration, a mapper agent computes the shear stress on each surface cell. Depending on the discretization used in the flow solver, different approaches can be used. If the flow grid is coarser than the spatial scale of the SMC model (for example, the radius of the cells), a proper algorithm must be used in order to determine which cells are in contact with the flow, then for each cell position, the shear stress is extrapolated from the closest boundary fluid nodes. Alternatively, if the flow discretization is sufficiently fine, allowing more fluid boundary nodes to interact with single cells, the shear stress on the cell surface can be calculated by averaging the values of the closest nodes.

Given the current drug concentrations and the cell configuration, the mapper agent approximates the concentration on each cell. As is the case for the shear stress approximation, the algorithm used depends on the grid size of the DD model. If the grid is sufficiently fine, with many lattice nodes per SMC, the concentration on a cell can be integrated. If a coarse DD grid is used, the concentration for each cell will be extrapolated using the closest nodes.

The single scale models, BF, SMC, DD, are implemented in different programming languages (FORTRAN90, C++, JAVA), wrapped as JAVA agents [11,12], and connected as shown in the Connection Scheme (Fig. 2). In the current setup we have 25 agents (3 kernels, 20 conduits and 2 mappers) which participate in the main computation of our prototype application. These can be distributed across multiple CPUs and machines to gain an advantage in computing speed.

5 Simulation Results

As a benchmark geometry for the 2D CxA model, we consider a vessel, of length 1.5 mm and width 1.0 mm, where two square struts of side length 90 μm have

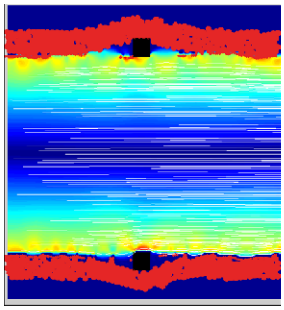


Fig. 3. Initial Condition for the CxA model, including initial cell configuration, equilibrated after deployment, and blood flow (the flow is shown in the lumen as streamlines, and the fluid shear stresses are color coded (red high, blue low)

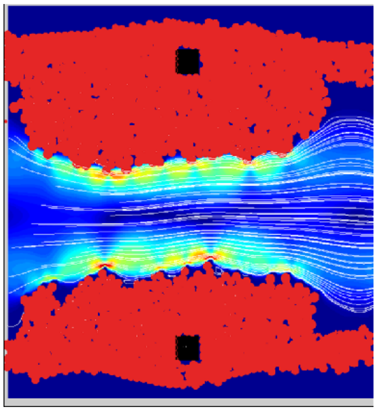


Fig. 4. Results with bare metal stent after 400 time steps (∼18 days). The flow is shown in the lumen as stream lines, and the fluid shear stresses are color coded (red high, blue low).

been deployed. The vessel wall has a thickness of 120 μm. Smooth muscle cells are generated with an average radius of 15 μm, densely packed inside the wall. To obtain the initial conditions based on the above geometry, an initial stress configuration compatible with the initial geometry must be provided. To do this, cells are stent deployment is simulated, iterating the structural SMC solver until a stationary state is reached. The initial cell configuration resulting from this procedure is shown in Fig. 3. The struts are shown in black, the SMC in red.

We have run the simulation for 400 time steps of the SMC model ($\Delta t = 1$ hour) for both a bare metal stent and a drug eluting stent. The final results are shown in Fig. 4 and 5 respectively. The preliminary results of the model demonstrate smooth muscle cell proliferation in response to injury caused by the stent deployment. Proliferation is modulated by flow, local mechanical stress, and

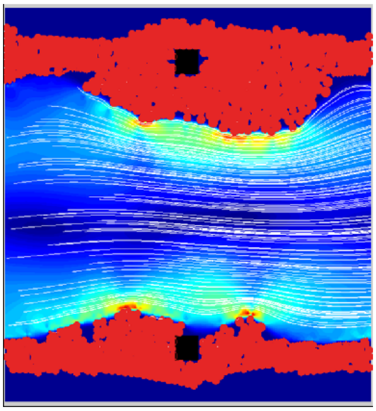


Fig. 5. Results with a drug eluting stent after 400 time steps (~18 days)

drug concentration (of particular note is the inhibitory effect observed when a drug eluting stent is simulated). The smooth muscle proliferation currently observed appears to share more in common with tumour cell growth, than the slower, more regulated hyperplastic response we would expect to see in a coronary artery following stent deployment. We are currently in the process of running more extended simulations and validating the model against a biological data-set obtained from stented porcine arteries. As parameters are tuned using archival quantitative data (from the literature), and experimental measurements, we hope to reproduce the positive correlation between neointimal growth rate and injury index. The next step will be to run a full 3D version of the model, enabling the influence of the stent geometry on the restenosis response to be investigated.

6 Conclusions

We have shown how Complex Automata methodology can be applied in a challenging multiscale model of in-stent restenosis. In particular, we described a conceptual multiscale model of in-stent restenosis, implementing the coupling of single scale algorithms for three different subprocesses with different time scales. The model has been realised employing a dedicated coupling library. The preliminary results demonstrate that the CxA model can be successfully implemented within this framework.

The individual models are at a relatively early stage of development and require further improvement. During the lifetime of the project we aim to achieve integration of a parallel flow solver (to achieve more detailed local hydrodynamics) and enhancement of the complexity of the SMC Agent rule-set. Moreover, the current 3-box CxA model might be improved further by including additional processes modelling thrombus formation and endothelial cell loss/re-growth. Validation of the improved and additional individual models using quantitative data derived from *in vivo* and *in vitro* experimentation will allow assessment of model accuracy, thus indicating the limitations of the models.

This first realisation of the coupled CxA is an important milestone on the journey towards a full multiscale model of in-stent restenosis, the subject of ongoing research.

Acknowledgments. This research is supported by the European Commission, through the COAST project (`www.complex-automata.org`, EU-FP6-IST-FET Contract 033664).

References

1. Evans, D., Lawford, P., Gunn, J., Walker, D., Hose, R., Smallwood, R., Chopard, B., Krafczyk, M., Bernsdorf, J., Hoekstra, A.: The Application of Multiscale Modelling to the Process of Development and Prevention of Stenosis in a Stented Coronary Artery. Phil. Trans. Roy. Soc. A 366, 3343–3360 (2008)
2. Morton, A., Crossman, D., Gunn, J.: The Influence of Physical Stent Parameters upon Restenosis. Pathologie Biologie 52, 196–205 (2004)
3. Kitano, H.: Computational Systems Biology. Nature 420, 206–210 (2002)
4. Sloot, P.M.A., Hoekstra, A.G.: Multiscale Modeling in Computational Biology. Accepted for publication in Briefings in Bioinformatics (2009)
5. Hoekstra, A.G., Lorenz, E., Falcone, J.-L., Chopard, B.: Towards a Complex Automata Formalism for Multi-Scale Modeling. Int. J. Mult. Comp. Eng. 5, 491–502 (2007)
6. Southern, J., Pitt-Francis, J., Whiteley, J., Stokeley, D., Kobashi, H., Nobes, R., Kadooka, Y., Gavaghan, D.: Multi-scale Computational Modelling in Biology and Physiology. Prog. Biophys. and Mol. Biol. 96, 60–89 (2008)
7. Hoekstra, A.G., Lorenz, E., Falcone, J.L., Chopard, B.: Towards a Complex Automata Framework for Multi-Scale Modeling: Formalism and the Scale Separation Map. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 922–930. Springer, Heidelberg (2007)
8. Hoekstra, A.G., Falcone, J.-L., Caiazzo, A., Chopard, B.: Multi-scale modeling with cellular automata: The complex automata approach. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 192–199. Springer, Heidelberg (2008)
9. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge University Press, Cambridge (1998)
10. Chopard, B., Falcone, J.-L., Razakanirina, R., Hoekstra, A.G., Caiazzo, A.: On the Collision-Propagation and Gather-Update Formulations of a Cellular Automata Rule. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 144–251. Springer, Heidelberg (2008)
11. Hegewald, J., Krafczyk, M., Tölke, J., Hoekstra, A.G., Chopard, B.: An Agent-Based Coupling Platform for Complex Automata. In: Sloot, P.M.A., et al. (eds.) ICCS 2008, Part II. LNCS, vol. 5102, pp. 227–233. Springer, Heidelberg (2008)

12. The COAST muscle (MUltiScale Coupling Library and Environment). Sources and Documentation, http://coast-dscl.berlios.de
13. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, Oxford (2001)
14. Stahl, B., Evans, D., Lawford, P. and Hose, R. (in preparation) (2009)
15. Hwang, C.-W., Wu, D., Edelman, E.R.: Physiological Transport Forces Govern Drug Distribution for Stent-Based Delivery. Circulation 104, 600–605 (2001)
16. Levin, A.D., Vukmirovic, N., Hwang, C.-W., Edelman, E.R.: Specific Binding to Intracellular Proteins Determines Arterial Transport Properties for Rapamycin and Paclitaxel. PNAS 101, 9463–9467 (2004)

A Mechano-Chemical Model of a Solid Tumor for Therapy Outcome Predictions

Sven Hirsch[1], Dominik Szczerba[2,1], Bryn Lloyd[1], Michael Bajka[3], Niels Kuster[2], and Gábor Székely[1]

[1] Department of Electrical Engineering, ETH, CH-8092 Zürich, Switzerland
[2] IT'IS Foundation, CH-8004 Zürich, Switzerland
[3] Division of Gynecology, University Hospital Zürich, Switzerland

Abstract. Experimental investigations of tumors often result in data reflecting very complex underlying mechanisms. Computer models of such phenomena enable their analysis and may lead to novel and more efficient therapy strategies. We present a generalized finite element mechano-chemical model of a solid tumor and assess its suitability for predicting therapy outcome. The model includes hosting tissue, tumor cells (vital and necrotic), nutrient, blood vessels and a growth inhibitor. At a certain time instant of the tumor development virtual therapies are performed and their outcomes are presented. The model is initiated with parameters either obtained directly from the available literature or estimated using multi-scale modeling. First results indicate the usefulness of multi-physics tumor models for predicting therapy response.

1 Introduction

According to the World Cancer Report, 12 million new cancer diagnoses are expected worldwide this year, and by 2010 it will be the leading cause of death. Better understanding tumor formation is of utmost social, economic and political importance, and finding more effective therapies may be regarded as one of the biggest challenges of our time. Computer simulation may bring new insights into the underlying mechanisms and may help to predict and optimize the effects of therapies. Simulation of such complex systems are computationally most demanding, yet the rapid development of hardware, especially emerging distributed parallel computing concepts, enable more and more realistic modeling of physiological systems. Tumor development has been studied extensively over the last three decades, reviews and exhaustive survey of the approaches can be found elsewhere [1,2] or more recently in [3]. A further discussion of the relevant literature may be found in our previous works [4,5]. We continue to rely on the established methods from solid mechanics and chemical engineering. The finite element method (FEM) is particularly suitable for describing strain-induced cellular responses, gaining acceptance as an influential player in simulating tissue development [6,7]. With such a representation we can account for mechanical tissue deformations induced by developing pathologies and capture its interplay with the chemical environment. We focus on benign, vascularized, solid tumor growth and present a significantly enhanced version of our previous model [8]. Extensions include:

1. more realistic boundary conditions for the growth factor and oxygen;
2. tumor compartmentalization;
3. deformation of the vascular system (displacement due to tumor growth);
4. effects of applying a growth inhibitor.

We do not only aim to simulate the tumor growth but also employ the model to study the effect of therapies. After describing the model we present its application to the simulation of embolization and the administration of angiostatin. Once precise experimental data are available for particular tumor types, we can integrate this quantitative knowledge into our framework and offer practical tools of clinical relevance.

2 Definition of the Model

The basis of our model is a set of mass and force balance equations. The mass transport of all constituents is modeled with reaction-convection-diffusion equations. Growth factor, endothelial cells, oxygen, and growth inhibitor are transported through the tissue, and may enter chemical reactions anywhere in the whole domain, which is shown schematically in Fig. 1. The components of this linked bio-chemo-mechanical model are described in the following equations where c_i denotes a concentration of a constituent, D_i is its diffusion coefficient and R_{ij} the reaction/source term. We assume *no flux* boundary conditions where the inflow from the environment is not accessible.

The growth factor is produced in hypoxic regions of the tumor and decaying naturally or through inhibition:

$$\frac{\partial c_1}{\partial t} = D_1 \nabla^2 c_1 + R_{11}(c_3) - R_{12}c_1 - R_{13}c_4 \tag{1}$$

$$R_{11}(c_3) = \begin{cases} R_{11}^0, & c_3 \geq t_1^{high} \\ R_{11}^0 + 5 \cdot R_{11}^0 \cdot \left(\frac{t_1^{high} - c_3}{t_1^{low} - t_1^{high}} \right), & t_1^{low} < c_3 < t_1^{high} \\ 6 \cdot R_{11}^0, & c_3 \leq t_1^{low} \end{cases} \tag{2}$$

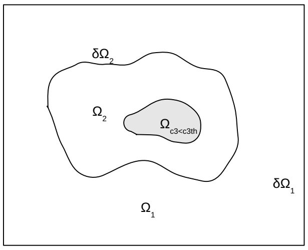


Fig. 1. Compartments of the model. The tumor (Ω_2) consisting of necrotic and viable part, is embedded into the host tissue (Ω_1). δ denotes the respective interfaces.

with *no flux* boundary condition of on $\delta\Omega_1$. The density of endothelial cells (EC) is formulated accordingly:

$$\frac{\partial c_2}{\partial t} = D_2 \nabla^2 c_2 + \nabla \cdot (c_2 \boldsymbol{u}) + \nabla \cdot (c_2 \boldsymbol{d}) \tag{3}$$

$$\boldsymbol{u} = \left(\frac{k_0\, k_1}{k_1 + c_1} \nabla c_1 \right) \tag{4}$$

where $\boldsymbol{d}$ is a mechanical displacement field and a *Dirichlet* boundary condition is applied on $\delta\Omega_1$. The oxygen concentration in the tissue obeys a reaction-diffusion equation of the form

$$\frac{\partial c_3}{\partial t} = D_3 \nabla^2 c_3 + R_{31}(c_3, c_2) - R_{32}(c_3) \tag{5}$$

with *no flux* boundary condition on $\delta\Omega_1$. Here (c_3, c_2) is the source term, which depends on the vasculature and blood flow. Angiostatin (AST) is provided externally as a drug that diffuses and decays naturally:

$$\frac{\partial c_4}{\partial t} = D_4 \nabla^2 c_4 - R_4 c_4 \tag{6}$$

with *no flux* boundary condition on $\delta\Omega_1$. We complete the above system of mass-balance equations by adding a force-balance equation describing the mechanical stress resulting from the evolution of the system as a consequence of tumor growth (Newton equilibrium equation)

$$\nabla \cdot \sigma + \boldsymbol{f} = 0. \tag{7}$$

σ is a second order tensor, $\boldsymbol{f}$ is the external force field. Here we pose a *no displacement* boundary condition everywhere on $\delta\Omega_1$ and an *initial strain* condition in Ω_2.

Growth Model: Healthy tissue is in a dynamic balance between proliferation and ceasing of cells, in consequence the number of cells remains nearly constant. The process of controlled cell death - apoptosis - is an integral part of the constant renewal of tissue in the natural cell cycle. The control mechanism is part of the homeostasis required by living organisms to maintain their internal states within certain limits. The down-regulation of apoptosis in tumorous tissue leads to an over-proportional growth. Apoptosis involves a series of biochemical events. The natural mechanism of apoptosis is controlled by intrinsic and extrinsic agents. Once initiated, apoptosis results in a characteristic cell shrinkage, blebbing, and DNA fragmentation. Such, the cell may be phagocytosed safely without exposing the tissue with potentially harmful intracellular debris. In contrary, necrosis is an uncontrolled death, e.g. due to hypoxia or toxic agents, and is characterized by an uncontrolled bursting of the cell membrane with a release of the constituents. This leads to a local inflammation and only partial resorption of the debris can be achieved. The overall density of cells inside the tumor N can be therefore decomposed into two compartments, vital N^+ and necrotic N^- cells:

$$N(t,x,y) = N^+(t,x,y) + N^-(t,x,y). \tag{8}$$

In our model, proliferation is constrained by the availability of oxygen [7,9] and space [10]. We account for tumor cells proliferation, apoptosis and necrosis. Necrotic cells formed from hypoxic tumor cells. g^+, g^{--}, g^- are the individual growth rates for each mechanism:

$$\text{growth: } g^+ = h^+_{c_3}(c_3) h^+_\sigma(\sigma) \frac{\ln 2}{T_2^+}, \tag{9}$$

$$\text{necrosis: } g^- = h^-_{c_3}(c_3) \frac{\ln 2}{T_2^-}, \tag{10}$$

$$\text{apoptosis: } g^{--} = \frac{\ln 2}{T_2^{--}}. \tag{11}$$

We neglected the pressure-growth dependency $h^+_\sigma = 1$. Populations N^+ and N^- are

$$\frac{\partial N^+(t,x,y)}{\partial t} = N^+(t,x,y)[g^+ - g^- - g^{--}], \tag{12}$$

$$\frac{\partial N^-(t,x,y)}{\partial t} = N^+(t,x,y)[g^-]. \tag{13}$$

All the equations introduced above are solved numerically with a generalized convection-diffusion-reaction solver relying on a standard finite element discretization. The domain has been mapped with triangular meshes, typically consisting of 50-100.000 triangles.

Tissue Mechanics: Tumor progression and regression is modeled as initial strain condition. Volumetric strain is:

$$\varepsilon_0(t) = \frac{dV(t)}{V(t)} = \frac{dN(t)}{N(t)} \approx \frac{\Delta N}{N(t)}. \tag{14}$$

The discrete form yields the population growth $\Delta N/N$ at timestep i:

$$\frac{\Delta N}{N_i} = \frac{(N^+_{i+1} + N^-_{i+1}) - (N^+_i + N^-_i)}{N^+_i + N^-_i} \tag{15}$$

and can also be negative, meaning cell degradation and removal. To solve the presented equations we rely on a commercial FEM package, Comsol Multiphysics. We verified this solver extensively with common benchmarks like flow past a cylinder, forced convection or a wall mounted bar sinking with gravity, where we found very good agreement with the reference solutions.

3 Example Application

In this section we present an application of our model to evaluate therapeutic effects. The realism of the simulation is obviously very sensitive to the parameter choice and determination of these parameters is a challenging part of tumor simulation. The parameter set of any specific tumor type is not fully known. Some of the parameters are difficult to access by experiment, others are being reported differently by different authors. It is not the aim of this study to take precise, well-defined parameters for one particular tumor type. Instead, for this study we want to prove the feasibility of the presented modeling approach and will rely on the literature or estimates to obtain the model parameters.

3.1 Model Parameters

Besides the set of parameters listed in Fig. 2 we will elucidate only the ones determined via additional considerations. The delivery of oxygen depends on the partial pressure difference in the blood and the tissue. It increases in hypoxic regions, while in regions with high concentrations only little oxygen is delivered:

$$R_{31}(c_3, c_2) = R_0(c_2) - R_p(c_2)(c_3 - c_3^0). \tag{16}$$

The reaction term parameters R_0, R_p and c_3^0 depend on factors such as the EC density c_2. The actual terms are derived in the next section.

O_2 consumption: The actual value of oxygen consumption [11,12] depends on many factors, including the tissue location, physical activity and altitude. For the consumption we assume a logistic term

$$R_{32}(c_3) = R_{sat} \frac{(c_3)^p}{(c_3)^p + \left(c_3^h\right)^p}, \tag{17}$$

Parameter	Description	Value	SI Units	Reference
E[R^2] =E[A}/pi	Expectancy value of R^2	1,40E-11	m^2	MacGabhann 2007
gamma	wall thickness to radius ratio	1,50E-01	-	MacGabhann 2007 (Kretowski, Liver: 0.1-0.2)
Po2^{blood}	Blood oxygen partial pressure	6,67E+03	Pa	Ji 2006 (Shibata 2001)
C_2^0	normal vessel density	2,50E+01	kg vessels / m^3 tissue	MacGabhann 2007
alpha^tissue	oxygen saturation in tissue	2,92E-07	m^3 O2/(m^3 tissue)/Pa	Ji 2006 (Ursino 0.024 ml O2/(ml tissue)/atm)
D_3	Diffusion coeff. of O2 in tissue	2,40E-09	m^2/s	Ji 2006, Salathe(1.5e-5 cm^2/s)
c_3^h	half saturation conc.	1,95E-05	m^3 O2 / m^3 tissue	Ji 2006 (Pcrit=0.5 mmHg)
c_3^0	median O2 conc	1,87E-03	m^3 O2 / m^3 tissue	Mayer 2008 (9 mmHg), Hoeckel 1991 (13mmHg in parous women)
p	shape of consumption curve	1,00E+00	-	Ji 2006
D_1	Diffusion coeff. of VEGF_164 in tissue	1,04E-10	m^2/s	estimated by MacGabhann 2007
D_1	Diffusion coeff. of TAF in tissue	5,00E-11	m^2/s	Anderson & Chaplain 1998
R_12	linear reaction coeff.	1,00E-06	1/s	unclear, Szczerba, 2008
R_11^0	TAF source factor	2,00E-13	mol/m3/s	Assuming 20kDa, unclear, refer to model in MacGabhann 2007
c_1 typical	"typical" VEGF level measured in serum (women, ovaries, uterus)	4,00E-03	g/m^3	Agrawal 1999, 2000
t_1^low	O2 threshold for hypoxic reaction	3,89E-05	m^3 O2 / m^3 tissue	Mac Gabhann 2007
t_1^high	O2 threshold for hypoxic reaction	7,78E-04	m^3 O2 / m^3 tissue	Mac Gabhann 2007
D_2	Diffusion coeff. of EC	1,00E-14	m2/s	Anderson 1998, Szczerba, 2008
k0	chemotactic coefficient	2,60E-04	m5/s/mol	Stokes 1990
k1	chemotactic coefficient	1,00E-07	mol/m3	Stokes 1990

Fig. 2. Summary of the model parameters. The references not covered by the bibliography of this paper can be found in an earlier publication [8].

where c_3^h is the concentration at which the reaction term reaches half maximum and p controls the shape of $R_{32}(c_3)$. This reflects the fact that the consumption is bounded by the amount of oxygen available but also saturates if oxygen is unlimited.

Oxygen Source Terms: The approximate form of the oxygen source terms can be estimated based on following observations. For a vessel segment of length L, radius R_s and wall thickness w_s the exchange of oxygen with the surrounding tissue depends on the oxygen gradient, respectively the partial pressure difference across the vessel wall (Fick's first law)

$$Q_{O_2} = dS\, D^w_{O_2} \alpha^w_{O_2} \left(P^{blood}_{O_2} - P^{tissue}_{O_2}\right) \frac{1}{w_s}, \tag{18}$$

where $dS = 2\pi\, R_s L$ is the surface of the vessel segment and $D^w_{O_2}$ is the diffusion constant in the vessel wall. Oxygen concentration in the tissue is proportional to the oxygen partial pressure according to Henri's law

$$P^{tissue}_{O_2} = c_3 / \alpha^{tissue}_{O_2}, \tag{19}$$

where the parameter $\alpha^{tissue}_{O_2}$ is the oxygen solubility. The term $\alpha^w_{O_2}(P^{blood}_{O_2} - P^{tissue}_{O_2})\frac{1}{w_s}$ is the oxygen concentration gradient across the vessel wall. For a short segment, the partial pressure in tissue $P^{tissue}_{O_2}$ and the blood $P^{blood}_{O_2}$ is approximately constant. In a given volume V the total oxygen delivered to the tissue is the sum of contributions from all segments within V

$$Q^V_{O_2} = 2\pi L \sum_k^N R_k D^w_{O_2} \alpha^w_{O_2} \left(P^{blood}_{O_2} - P^{tissue}_{O_2}\right) \frac{1}{w_k}. \tag{20}$$

The EC density c_2 includes the vessel wall and lumen, within the volume V it is

$$c_2 = \frac{1}{V} \sum_k^N L \cdot R_k^2\, \pi = \frac{L}{V} \sum_k R_k^2\, \pi. \tag{21}$$

Solving (21) for the segment length L and inserting the result in (20), gives a relation between the oxygen flow $Q^V_{O_2}$ and EC density. Under the assumptions of constant blood oxygen partial pressure and constant wall thickness to radius ratio γ, the terms can be simplified further. Finally, the source terms can be deduced by dividing the oxygen flow by the volume V

$$R_{31}(c_3, c_2) = \frac{Q^V_{O_2}}{V} = c_2 \frac{2/\gamma \cdot D^w_{O_2} \alpha^w_{O_2} \left(P^{blood}_{O_2} - P^{tissue}_{O_2}\right)}{\sum_k R_k^2 / N} \tag{22}$$

We recognize the term $\sum_k R_k^2/N$ as the expected value $E[R^2]$.

$$R_{31}(c_3, c_2) = c_2 \frac{2/\gamma \cdot D^w_{O_2} \alpha^w_{O_2} P^{blood}_{O_2}}{E[R^2]} - c_2 \frac{2/\gamma \cdot D^w_{O_2} \alpha^w_{O_2} / \alpha^{tissue}_{O_2} \cdot c_3}{E[R^2]} \tag{23}$$

$$= R_A c_2 - R_B c_2 c_3 \tag{24}$$

It is easy to verify that the derived source term corresponds to the generic form proposed in (16) with appropriate selection of the terms R_0 and R_p.

3.2 Initialization and Growth

The simulations are initiated with a small cluster of tumor cells in the center of the host tissue surrounded by vessels on each side. The corresponding high EC density is visible in Fig. 4 as a decaying gray shadow gradient at the box walls. The tumor promotes directed vessel growth via secretion of tumor angiogenic factors (TAF), leading to EC migration from the adjacent parent vessels. In this vascular phase the tumor expands virtually unbounded and will eventually cause physiological problems due to compression of the surrounding tissue. The EC density increase the corresponding vessels' diameter and wall thickness in time, which in turn modulates their oxygen delivery rate. For large vessels, blood flow increases and diffusion through the wall decreases. For capillaries it is the contrary: most of oxygen exchange is realized here, but the net mass flux is small. To obtain the distributions of vessel diameters we solve a *Dirichlet* problem inside the domain by fixing the vessel diameters at the boundaries to the expected diameters (feeding arterioles, 50 μm). The equation parameters are determined experimentally to achieve capillaries of about 5 μm radius in the center of the domain. The unregulated tumor growth is presented in Fig. 4 left (EC) and Fig. 3 (oxygenation map). Non-symmetrical compartments develop despite the initial boundary symmetry. EC density is realized with a typical capillary buildup in the center of the host tissue in form of a frequently observed vascular capsule.

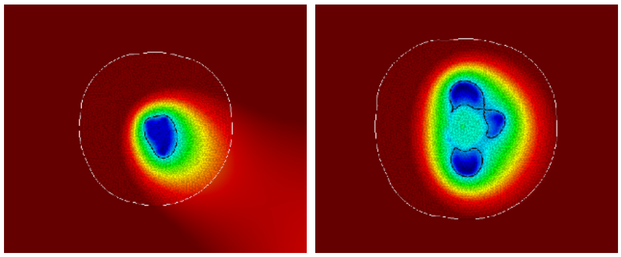


Fig. 3. Examples of compartment formation within the tumor (white outline with mesh) of necrotic and viable tissue for different necrosis thresholds. The oxygenation map is color coded, the inner areas denote oxygen isocontours of the threshold level.

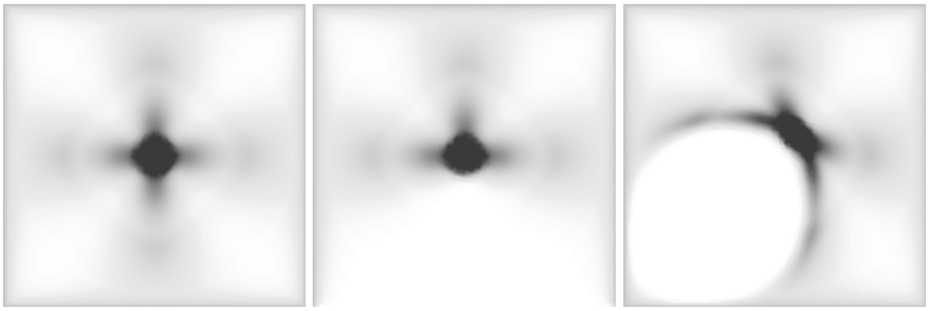


Fig. 4. EC density within the host tissue. Left: unregulated growth; center: embolization of the lower vessel; right: tumor after angiostatin infusion.

Four branches of vessels are clearly visible, connecting the tumor to the feeding arteries on the periphery. The tumor itself is penetrated by a dense network of capillaries, as often actually observed.

Growth inhibition through embolization: Embolization is a minimally-invasive procedure involving a selective occlusion of blood vessels, by introducing an artificial embolus (coil, particle, foam, plug). The purpose of embolization is to permanently block the blood flow to the tumor, leading to immediate tissue hypoxia and eventually necrosis. In our simulation the tumor grows initially unbounded, supported by the accompanying angiogenesis for 4 months. The virtual embolization procedure is performed by disabling the lower feeding artery. We implement it by replacing the *Dirichlet* boundary condition by a *non-penetration* condition. Note, that we do not remove any existing daughter vessels that sprouted off. This relatively well approximates the fact that only blood supply is eliminated, but neither the blood vessels nor endothelial cells. The EC density concentration 6 months after treatment in Fig. 4 (center) shows an asymmetric appearance. The vessel density in the lower half is greatly reduced, which corresponds to a ceasing

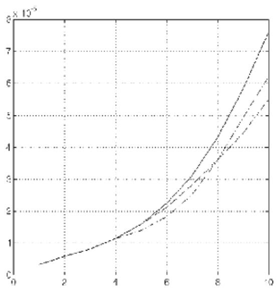


Fig. 5. Development of total tumor mass under different conditions. Solid line: no treatment; dashed line: Angiostatin treatment; dot dashed line: embolization.

of the respective feeding vessel. The capillary core is still maintained, and shows only slight signs of asymmetry in the lower part. Obviously the capillaries still sufficiently supply the tumor with oxygen, such that the vessel network can be maintained through preexisting anastomosis. As the daughter vessels were not removed, they are still a part of a network and continue to deliver oxygen. In Fig. 5 we can, however, observe that the total mass of the tumor is somewhat lower than in the unregulated case.

Growth inhibition through Angiostatin: Next we test the infusion of angiostatin (AST), a potent inhibitor of angiogenesis. There are several ways angiostatin influences the tumor and the underlying mechanisms are not fully understood [13,14]. In the study we introduce angiostatin locally in the 4th month of tumor development and continue its administration till the end of month 10. This inhibitor neutralizes the growth factor directly as described by (2). We apply AST off-center in the lower left quadrant and not inside the tumor, to prevent the vessels from reaching the tumor at all. In Fig. 4 (right) we notice a strongly asymmetric EC density distribution, where the AST supplied area is excluded from any vessel growth. The local concentration of AST does not actually prevent the vessels from connecting to the tumor. Instead, the EC density builds clearly defined feeding vessels around the AST supplied area. The EC density accumulates outside the AST application region and is able to form a dense capillary capsule. We were not able to find data in the literature for the reaction rate between AST and the TGF, and it was arbitrarily set 1:1, justified to some extent by the similar molecular weights of the two factors.

4 Discussion and Outlook

The results from our extended model are in a reasonable qualitative agreement with commonly available findings. The improved modeling of EC added further physiological realism, since we managed to model the capillary penetration of the tumor and are now able to achieve tumor compartmentalization as seen in Fig. 3. This approach generates reasonable results, which make it a viable alternative to the expensive explicit modeling of the vessels. The origin of asymmetric phenotypes from symmetric boundary conditions we will consider closely with refined meshes. We will further extend the scope of our model, to cover also other aspects like thermal effects, hormonal, radiation or chemotherapy. The tumor model is generally difficult to validate since many of the parameters are unknown or carry large measurement errors. It is the most difficult challenge to quantify these effects on the cellular level, which is the reason why the simulation outcome can not be currently validated against the physiological findings. After all the development of the tumor may be assessed on the macroscopic level, comparing the tumor regression to *in vivo* diagnostic observation (CT, US, MRI). We are convinced of the great value of the computational model once parameters are available from measurements. Here we are especially interested in verifying mechanical effects as it is one of the major benefits of relying on FEM methods for the simulation.

Acknowledgments. This work is part of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me), supported by the Swiss National Science Foundation.

References

1. Araujo, R.P., McElwain, D.L.S.: A history of the study of solid tumour growth: the contribution of mathematical modelling. Bulletin of Mathematical Biology 66(5), 1039–1091 (2004)
2. Byrne, H.M., Alarcon, T., Owen, M.R., Webb, S.D., Maini, P.K.: Modelling aspects of cancer dynamics: a review. Philos. Transact. A Math. Phys. Eng. Sci. 364(1843), 1563–1578 (2006)
3. Bellomo, N., Li, N., Maini, P.: On the foundation of cancer modelling: Selected topics, speculations, and perspectives. Mathematical Models and Methods in Applied Sciences 18(4) (2008)
4. Szczerba, D., Székely, G., Kurz, H.: A multiphysics model of capillary growth and remodeling. Simulation of Multiphysics Multiscale Systems (2006)
5. Lloyd, B.A., Szczerba, D., Rudin, M., Székely, G.: A computational framework for modelling solid tumour growth. Philos. Transact A. Math. Phys. Eng. Sci. (July 2008)
6. Gordon, V.D., Valentine, M.T., Gardel, M.L., Andor-Ardo, D., Dennison, S., Bogdanov, A.A., Weitz, D.A., Deisboeck, T.S.: Measuring the mechanical stress induced by an expanding multicellular tumor system: a case study. Experimental Cell Research 289(1), 58–66 (2003)
7. Graziano, L., Preziosi, L.: Mechanics in tumor growth. In: Mollica, F., Rajagopal, K., Preziosi, L. (eds.) Modelling of Biological Materials, pp. 267–328. Birkhäuser, Basel (2007)
8. Szczerba, D.: A multiphysics model of myoma growth. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part II. LNCS, vol. 5102, pp. 187–196. Springer, Heidelberg (2008)
9. Anderson, A.R., Chaplain, M.A.: Continuous and discrete mathematical models of tumor-induced angiogenesis. Bulletin of Mathematical Biology 60(5), 857–899 (1998)
10. Chaplain, M.A.J., Graziano, L., Preziosi, L.: Mathematical modelling of the loss of tissue compression responsiveness and its role in solid tumour development. Math. Med. Biol. 23(3), 197–229 (2006)
11. Salathe, E.P., Xu, Y.H.: Non-linear phenomena in oxygen transport to tissue. Journal of Mathematical Biology 30(2), 151–160 (1991)
12. Ji, J.W., Tsoukias, N.M., Goldman, D., Popel, A.S.: A computational model of oxygen transport in skeletal muscle for sprouting and splitting modes of angiogenesis. J. Theor. Biol. 241(1), 94–108 (2006)
13. Sim, B.K., MacDonald, N.J., Gubish, E.R.: Angiostatin and endostatin: endogenous inhibitors of tumor growth. Cancer Metastasis Rev. 19(1-2), 181–190 (2000)
14. Folkman, J.: Role of angiogenesis in tumor growth and metastasis. Semin. Oncol. 29(6 Suppl 16), 15–18 (2002)

Simulating Individual-Based Models of Epidemics in Hierarchical Networks

Rick Quax, David A. Bader*, and Peter M.A. Sloot

University of Amsterdam, Faculty of Sciences,
Science Park 107 1098 XG Amsterdam, The Netherlands

Abstract. Current mathematical modeling methods for the spreading of infectious diseases are too simplified and do not scale well. We present the Simulator of Epidemic Evolution in Complex Networks (SEECN), an efficient simulator of detailed individual-based models by parameterizing separate dynamics operators, which are iteratively applied to the contact network. We reduce the network generator's computational complexity, improve cache efficiency and parallelize the simulator. To evaluate its running time we experiment with an HIV epidemic model that incorporates up to one million homosexual men in a scale-free network, including hierarchical community structure, social dynamics and multi-stage intranode progression. We find that the running times are feasible, on the order of minutes, and argue that SEECN can be used to study realistic epidemics and its properties experimentally, in contrast to defining and solving ever more complicated mathematical models as is the current practice.

1 Introduction

Faithful simulations of epidemics in population networks require explicit modeling of a large number of static and dynamic properties of the network and the epidemic. However, current research often performs rigorous mathematical studies of non-representative simplifications. Examples of such network properties are degree distribution, community structure, assortativity, node and edge types, edge weights and temporal variance; properties of epidemics include infection stage progression, infectiousness, drug efficacy and immunity. In particular, epidemics are typically simulated using standard mean-field approximations or master equations extended with one or a few of these properties, but extending these to more representative models is difficult.

It is widely accepted that current mean-field approximations and master equations of epidemics are unrealistic and that more social and epidemiological details should be considered [1,2,3,4], so various biological and social systems are thought of as networks with various complex properties [5,6,7,8,9]. At present, however, epidemic studies using such complex networks focus on only one or a few such properties, such as degree correlation [10,11,12,13,14,15,16], topology [17,16,18,19,20,21],

* College of Computing, Georgia Institute of Technology, USA.

edge weights [22,21], temporal variance [23], and epidemic dynamics [24,25]. At the other extreme and similar in spirit to our work is EpiSims [26], which uses detailed infrastructure and traffic data of a single city and simulates on the scale of seconds; however it does not extend easily to other and larger populations.

In this paper we present the Simulator of Epidemic Evolution in Complex Networks, or SEECN[1], with which detailed individual-based models can be simulated experimentally, based on available data. In SEECN, nodes and edges are organized hierarchically and have arbitrary properties that dictate the temporal evolution of the network and the epidemic. This evolution is driven by a set of *dynamics operators* that can be parameterized independently and in terms of the node and edge properties; for example, sexual relationships (edges) form and break depending on both nodes' gender and age. Their hierarchical organization reduces the model complexity and reflects how network data is typically available; for example, commuter traffic statistics may be available per province within a country, and per city within a province.

A consequential concern with such an expressive approach is running time, which is a challenge to optimize since scale-free networks often exhibit low degrees of locality and memory access patterns are not known a priori. We show how SEECN reduces computational complexity by exploiting hierarchical structure, improves cache efficiency by choosing an appropriate data structure and buffering edge traversals, and achieves parallelization with load balance. Our experiments show that these improvements reduce the running time to the extent that the simulator is practical even for detailed and highly entropic models.

This paper is organized as follows. In Section 2, SEECN is defined at a high-level with a minimum of implementation details, implying its expressiveness. Then, in Section 3, we discuss its algorithms and improvements. The experiments are presented in Section 4, and their results are shown and discussed in Section 5. Finally, we conclude in Section 6.

2 The SEECN

A simulation is boot-strapped by generating a representative network $G = (V, E)$, after which temporal *dynamics* are iteratively performed that drive the epidemic or change the network. See Fig. 1. We define dynamics as processes that change the network state in some way. All dynamics (including the network generator) are implemented as separate *dynamics operators* $\mathcal{D}_i$, or operators, which can be parameterized independently or reuse existing parameters. A model is a set of parameterized operators $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_d\}$.

To differentiate dynamics in a heterogeneous population, we augment each node $x \in [0, \|V\|)$ and edge $(x, y) \in E$ with a property vector $v_x \in \mathcal{V}$ or $w_{(x,y)} \in \mathcal{W}$, respectively, and parameterize operators in terms of these vectors. Each value $v_x[i]$ (or $w_{(x,y)}[j]$) of such a vector is one of a predefined set $\mathcal{V}[i]$ ($\mathcal{W}[j]$) of possible values for a specific node property i, such as gender or infection stage, or edge property j, such as type of relationship. Then, an operator implements a

[1] Pronounce as "season".

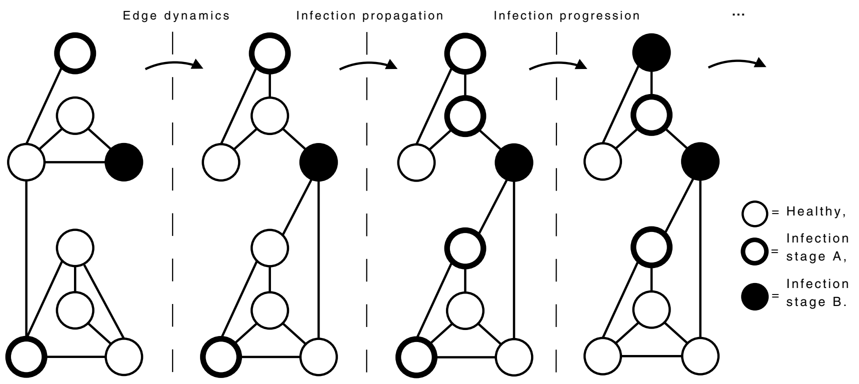


Fig. 1. Dynamics operators are applied sequentially to a network in one time step. The network has two communities, each with more internal than external edges.

transition probability matrix of $\|\mathcal{V}\| \times \|\mathcal{V}\|$ or $\|\mathcal{V}\|^2 \cdot \|\mathcal{W}\| \times \|\mathcal{V}\|^2 \cdot \|\mathcal{W}\|$ elements, essentially defining first-order stochastic differential equations for all network state[2]. Adding and removing nodes[3] and edges are special cases.

We use a separate operator for each type of dynamics, simplifying model specification. In this way, semantically different dynamics can be independently parameterized and turned on or off. In practice, each operator usually 'ignores' most of the parameter space; for instance, infection propagation changes only the receiving node's state, not that of the edge or the originating node.

In particular, the network generator $\mathcal{G}(h, p, q, N)$ is a special operator that is executed first and once. The first parameter is a network *recipe* $h : [0, N)^2 \mapsto [0, 1]$, which specifies the probability of each possible edge to be generated. Second, $p : \mathcal{V} \mapsto [0, 1]$ and $q : \mathcal{W} \mapsto [0, 1]$ specify the prior probability of any node or edge having a particular state. Lastly, N is the number of nodes.

The recipe is constrained as a modular network with hierarchical organization, or *hierarchy*, where the network is iteratively partitioned into subhierarchies. All nodes within a particular subhierarchy have equal connection probabilities to nodes of all other subhierarchies not contained by or containing the former. In particular, we define a *community* such that node x and y belong to the same community iff $\forall_z \; P((x, z) \in E) = P((y, z) \in E)$. In the worst case, the communities are precisely the lowest subhierarchies (which are not partitioned further), however in some cases multiple subhierarchies may be combined. Ultimately, a typical network's recipe is simplified with additional a priori statistical structure that we can exploit in our algorithms and analyses. Note that a single partition into N subhierarchies yields a recipe with no additional structure.

In addition to the benefit of a simplified recipe, we use hierarchical organization because many population networks appear hierarchical, and data is

[2] A *state* of a node or edge is a specific instance of its property vector; the network state is the combined state of its nodes and edges.

[3] Currently we only *replace* nodes, keeping network size constant.

often available in this way. For example, one may use travel statistics between provinces in a country, and between cities in a province. In this way, in the absence of individual data, each contact's probability can be estimated by traversing the hierarchy and multiplying the appropriate connection probabilities.

Finally, if network G_0 is an instance of a recipe, it changes over time step $i \geq 0$ as G_i when iteratively applying d dynamics operators, as

$$\begin{aligned} G_{i+1} &= \mathcal{D}_1\left(\mathcal{D}_2\left(\ldots \mathcal{D}_d\left(G_i\right)\ldots\right)\right) \ , \\ G_0 &= \mathcal{G}\left(h, p, q, N\right) . \end{aligned}$$

(Here, $\mathcal{D}(G_i)$ is shorthand for applying the operator to all existing nodes and edges at time step i.) Any statistic can then be calculated from the succession $(G_i)_i$, e.g., infection incidence, prevalence or treatment uptake.

SEECN's current implementation has some limitations. Most prominently, the hierarchy is constrained to that of "Kronecker graphs" [27] because of its good qualitative correspondence to real networks in absence of more detailed parameters that enable defining a recipe. This results in a recursive partitioning into two subhierarchies with equal connection probabilities at all levels. Further, edge probabilities are independent of property vectors; although this shortcoming can be significant, it can be partly overcome by changing the parameters for disease propagation.[4] Lastly, nodes are only removed or added, but do not migrate between communities.

3 Algorithmic Improvements

A primary concern with detailed individual-based simulations is running time. Many real networks are approximately scale-free [28], but such networks are notoriously difficult to partition and access patterns are highly irregular. Moreover, multiple simulation runs must be combined to obtain statistical significance, and typical applications include parameter-searching and impact evaluation of parameters. Therefore, a simulator should run fast in order to be useful.

For ease of presentation, we characterize operators as one of the following *primitives* that compute in $\mathcal{O}(N)$, $\mathcal{O}(|E|)$, and $\mathcal{O}(N^2)$ instructions[5], respectively.

Node operators visit all nodes, e.g., progression and removal.
Edge operators visit all existing edges, e.g, infection propagation.
Recipe operators visit all possible edges, e.g., network generation.

Our algorithmic improvements focus on computational complexity, cache efficiency and parallelization. Firstly, recipe operators dominate computation time and become a bottleneck for larger graphs even though such operators are cache efficient. The second bottleneck is due to cache misses, because the simulator performs many random memory accesses per little computation. Finally, an obvious but non-trivial improvement is parallelization.

[4] For instance, if unsafe sex with an HIV-infected person is less frequent, its infection probabilities are reduced.

[5] We distinguish computational complexity, in terms of instruction count, from running time complexity, which includes communication and memory access overhead.

3.1 Improving Computational Complexity

For recipe operators, we can exploit regularity imposed by hierarchical recipes. In contrast, the complexities of node and edge operators cannot be improved, assuming that every existing node and edge is relevant for the evolution of the epidemic. The following observation holds even if edge probabilities depend on property vectors since there is a constant number of possible states.

Many ($\omega(1)$) entries of a recipe are equal if $C \in o(N)$, where C is the number of communities. Since edge probabilities are equal for all nodes within a community, recipe operators compute in terms of communities instead of nodes. For example, to generate edges for a node its degree is drawn from a suitable distribution, since each community's contribution to a node's degree is binomially distributed. For larger communities these approach Poisson distributions that sum up. Then, each edge's neighbor node id is selected by first selecting a community id $[0, C)$ and then a random node id within that community.

Thus, recipe operators require $\mathcal{O}(NC)$ instructions. In worst-case, though, $C \propto N$ in which case it remains $\mathcal{O}(N^2)$; for Kronecker graphs, $C = \log_2 N$ [29].

3.2 Improving Cache Efficiency

Random memory access dominates running time due to visiting and traversing edges, both of which require special attention [30] and are discussed in turn. Node and recipe operators can visit node structures and random number generators in order, so this section focuses on edge operators.

Network dynamism further constrains the choice of data structure and suggests the classic adjacency matrix (see left of Fig. 2). To remain efficient, all memory should be pre-allocated and not be (de)allocated constantly, which would fragment memory. The adjacency matrix stores edges contiguously per node and reserves equal capacity $0 < D \leq N$ for each node. Visiting all edge sides of one node is in cache. However, in scale-free networks only a small fraction of nodes uses its full capacity, in which case visiting the first edge side of the next node is (almost) always a cache miss. Thus, the adjacency matrix incurs an expected $\langle M \rangle = N$ cache misses per node.

An alternative is the Jagged Diagonal Storage (JDS) [31] which stores node x's ith edge in the ith of D arrays, at element x of N (see right of Fig. 2). In JDS, all edge sides are visited per such array, and a cache miss occurs only when some node's ith edge is accessed while $B \geq 1$ previous nodes had less than i edges. To minimize this probability we renumber nodes on expected degree.

We can show that, for *visiting* all edge sides, JDS incurs fewer expected cache misses $\langle M \rangle$ than the adjacency array, as follows. We assume that a cache miss occurs if two memory accesses are separated by B or more edge structures. Node x has degree d with probability $f_x(d)$, and $F_x(d) = \sum_{i=0}^{d} f_x(i)$. The probability $P(M_x \geq m)$ of node x incurring at least $0 < m \leq D$ cache misses is then

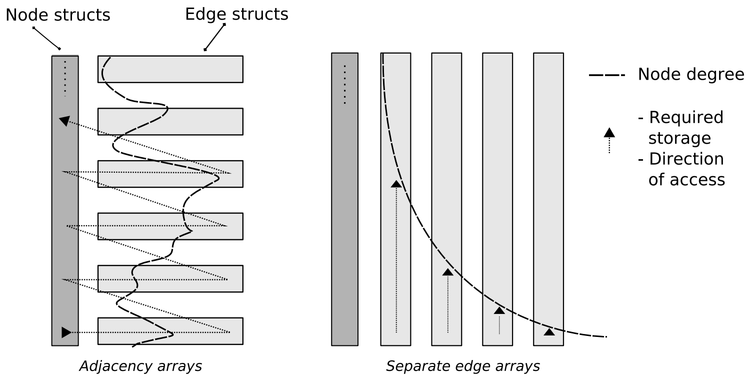


Fig. 2. A comparison of memory access patterns between the adjacency array and JDS for a scale-free network

$$
\begin{aligned}
P(M_x \geq m) &= \sum_{d=1}^{D} f_x(d) \cdot \prod_{y=1}^{L} F_{x-y}(d-m) \\
&\leq \sum_{d=1}^{D} f_x(d) \cdot F_{x-1}(d-m) && (L=1) \\
&\leq \sum_{d=1}^{D} f_x(d) \cdot F_x(d-m) \ < \frac{1}{2} && (\forall_{d'} \ F_x(d') \geq F_{x-1}(d'))
\end{aligned}
$$

(For binomial f_x.) In other words, most nodes will incur no cache miss at all. Although this does not prove that $\langle M \rangle < 1$ for all possible sets of parameters, we have been unable to find such sets experimentally. Computed using the above, $\langle M \rangle \approx 0.6$ for the simulations in the next section.

The second source of cache misses is due to *traversing* edges, which we reduce by queuing and sorting *updates*. Operators in SEECN queue updates, which dictate state changes for a neighboring node. Eventually the updates are sorted on node identifier and performed in order, splitting the traversal problem into a sorting part and an edge visiting part. For improved efficiency, the updates of multiple independent operators may be combined.

3.3 Parallelization

Because edge traversals dominate running time for large graphs, and usually $|E| \in \Omega(N)$, we partition the network with respect to edges. In contrast, node operators are relatively inexpensive because they access all nodes in order. Moreover, node structures must already be accessed for edge operators, e.g., to update a node's degree after edge removal or addition.

Therefore we partition the network at the granularity of nodes, in contiguous ranges, while balancing the expected total number of edges to be handled by each process. Although SEECN could exploit the hierarchical community structure to partition the network more optimally, the current partitioning algorithm is simplified by assuming no assortativity beyond that in expected degree; in fact, Kronecker is such a generator [29]. This assumption implies that a node x with expected degree $\langle d_x \rangle$ expectedly incurs $\langle d_x \rangle / \langle d_y \rangle$ more interprocess edges than node y, and therefore the optimal partition is to balance the expected number of edges to handle by any process i out of p. The probability that an edge connects two nodes of different processes is $(p-1)/p$.

In terms of implementation, few changes need be made. Most prominently, each process' update queue (Section 3.2) must be split into p queues, one for each process; at intermediate steps the queues can easily be transferred at once, benefiting from high communication bandwidth. Exploiting high bandwidth is important because the size of a queues scales as $\mathcal{O}(|E|)$ for increasing N.

4 Experiments

In this section we perform benchmark experiments using one, four and sixteen processes. The epidemic model is fairly complex and represents HIV in a population of homosexual men. This model assumes a hierarchical network with power-law exponent 1.6, and classifies nodes as healthy, acute, (un)treated asymptomatic, or (un)treated AIDS, each of which have distinct infectiousness, life expectency and duration of relationships. The details are presented elsewhere [29], where good qualitative correspondence with historical data was found.

We experiment with three variants of SEECN to evaluate the impact of cache efficiency and load balance. The first algorithm (`cached` or `C`) queues updates but does not renumber the nodes on descending degree (Section 3.2) and does not sort the queues. As variations to `cached`, the second algorithm (`load-balanced` or `LB`) renumbers the nodes and the third (`sorted` or `S`) also sorts the queues.[6] The latter two algorithms partition the graph into equal expected edge counts per process, whereas the `cached` algorithm must resort to balancing the number of nodes due to the irregular distribution of edges over nodes.

The experiments are run on a Linux cluster of 680 dual-core Intel Xeon nodes at 3.4 GHz. We allocate one process per compute node (homogenizing communication overhead) and use TCP/IP over Infiniband. We implement SEECN in C++, parallelize it with MPI, and compile it with GCC 4.1.2 and MPICH2 1.0.8. SEECN's code is not hand tuned and uses the standard STL sorting algorithm.

5 Results

The sequential experiments show the impact of cache efficiency of both visiting (`load-balanced`) and traversing (`sorted`) edges, for increasing N. The results are shown in Fig. 3. We could not experiment with network sizes of over 2^{20} because a single process would run out of memory.

[6] Particularly, we do not analyze the impact of using the JDS instead of the adjacency array, because only JDS is implemented.

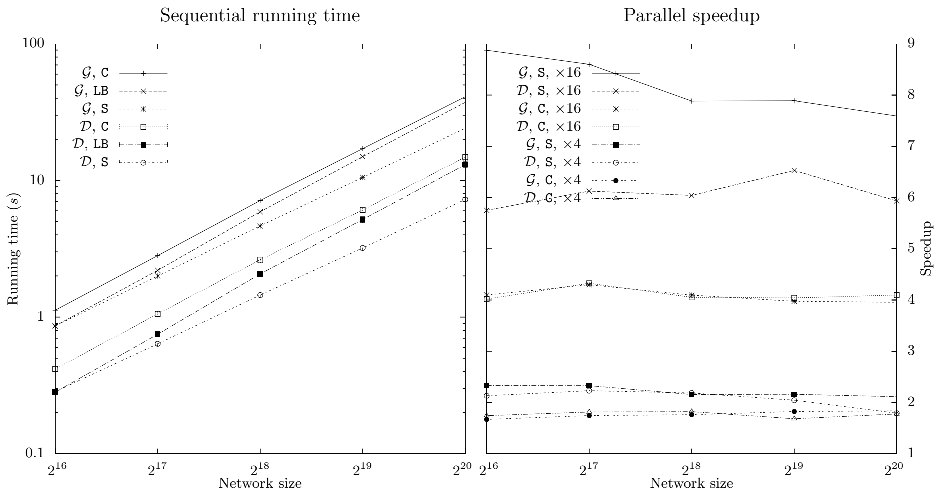


Fig. 3. The sequential running times (on logarithmic scale) of generating the network ($\mathcal{G}$) and performing all other dynamics in a single time step ($\mathcal{D}$), and the parallel speedups of both using 4 and 16 processes. All data points are averages of five runs, and each run had 100 time steps of dynamics (standard error is too small to show).

Firstly, the results suggest indeed that the running time scales as $|E|$. The lorarithmic running time plots of the `sorted` generator and dynamics operators have slopes of 2.26 and 2.30. The theoretically expected slope for the logarithmic edge count is 2.3 (not shown) because $|E| \propto 2.3^{\log_2 N}$ [29], in the absence of a maximum degree. It is also interesting that the fraction of edge visits and traversals that are cache efficient without sorting queues decreases with increasing network size, because both slopes for the `LB` algorithm are about 2.45.

Secondly, the performance benefit of renumbering nodes and visiting them in order is significant for smaller graphs ($N \leq 2^{16}$), for which sorting queues has no benefit. However, the number of updates per time step grows as $|E|$, so the average number of updates per node increases, whereas the number of initial visits (i.e., not caused by an update) remains constant. Consequently, for larger graphs ($2^{16} < N \leq 2^{20}$) the benefit of cache efficient initial visits diminish while the benefit of sorting updates and performing them in order increases.

For the parallel case we show the speedup of the `cached` and `sorted` algorithms over the same range as for the sequential case in Fig. 3. Clearly, balancing the number of nodes is less efficient than balancing edge counts. For an increasing number of processes, `sorted`'s efficiency curves are about 0.5 and almost constant; in contrast, `cached`'s efficiency curves drop from 0.4 to 0.25.

The speedups remain roughly constant over our range of N, which would be the case if both processing and communication time would be dominated by the number of updates and therefore by $|E|$. Communication overhead that is sublinear in $|E|$ in the absence of additional assortativity is not possible (without load imbalance), so we consider our parallelization efforts successful.

6 Conclusion

In this paper we show that detailed, individual-based models of epidemics can indeed be simulated in reasonable running time; in particular, we discuss and implement algorithmic improvements and evaluate the impact of buffering edge traversals, sorting these traversals, and parallelizing with balanced numbers of nodes or edges. For a representative epidemic model we find a constant sequential running time reduction of almost factor 2, and a constant parallel eficiency of roughly 0.5. As a result, a detailed simulation of HIV among one million persons in a hierarchical and scale-free network over 25 years (100 time steps) takes two minutes using 16 processes. We conclude that for simulating epidemics, experimentation can be an expressive and convenient alternative to mathematical modeling, and that parameter searching and impact evaluation are feasible.

Acknowledgments. We thank Breanndán Ó Nualláin and Dung Manh Chu for their help with the cache efficiency derivations, and Richard Vuduc for pointing out the existence of the Jagged Diagonal Storage. Bader's research is supported in part by NSF Grant CNS-0614915, Pacific Northwest National Laboratory's Center for Adaptive Supercomputing Software and by MIT Lincoln Laboratory. This research is supported by European ViroLab Grant INFSO-IST-027446.

References

1. Keeling, M.: The implications of network structure for epidemic dynamics. Theoretical Population Biology 67(1), 1–8 (2005)
2. Pastor-Satorras, R., Vespignani, A.: Epidemic dynamics in finite size scale-free networks. Phys. Rev. E 65(3), 035108 (2002)
3. Keeling, M., Eames, K.: Networks and epidemic models. Journal of the Royal Society Interface 2 (November 2005)
4. Parham, P.E., Ferguson, N.M.: Space and contact networks: capturing the locality of disease transmission. Journal of The Royal Society Interface 3, 483–493 (2006)
5. Strogatz, S.H.: Exploring complex networks. Nature 410(6825), 268–276 (2001)
6. Wuchty, S., Ravasz, E.: lászló Barabási, A.: The architecture of biological networks. In: Complex Systems in Biomedicine. Kluwer Academic Publishers, Dordrecht (2003)
7. Barabási, A.L., Oltvai, Z.N.: Network biology: understanding the cell's functional organization. Nature Reviews Genetics 5(2), 101–113 (2004)
8. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.U.: Complex networks: Structure and dynamics. Physics Reports 424(4-5), 175–308 (2006)
9. da Costa, F.L., Rodrigues, F.A., Travieso, G., Boas, P.R.V.: Characterization of complex networks: A survey of measurements. Advance. Advances In Physics 56, 167 (2007)
10. Sloot, P.M.A., Ivanov, S.V., Boukhanovsky, A.V., van de Vijver, D.A.M.C., Boucher, C.A.B.: Stochastic simulation of HIV population dynamics through complex network modelling. Int. J. Comput. Math. 85(8), 1175–1187 (2008)
11. Pastor-Satorras, R., Vespignani, A.: Epidemics and immunization in scale-free networks. In: Bornholdt, S., Schuster, H.G. (eds.) Contribution to Handbook of Graphs and Networks: From the Genome to the Internet. Wiley-VCH, Berlin (2002)

12. Pastor-Satorras, R., Vespignani, A.: Epidemic dynamics and endemic states in complex networks. Physical Review E 63, 66117 (2001)
13. Moreno, Y., Pastor-Satorras, R., Vespignani, A.: Epidemic outbreaks in complex heterogeneous networks. The European Physical Journal B - Condensed Matter and Complex Systems 26(4), 521–529 (2002)
14. Zhou, C., Kurths, J.: Hierarchical synchronization in complex networks with heterogeneous degrees. Chaos: An Interdisciplinary Journal of Nonlinear Science 16(1), 015104 (2006)
15. Barthelemy, M., Barrat, A., Pastor-Satorras, R., Vespignani, A.: Dynamical patterns of epidemic outbreaks in complex heterogeneous networks. Journal of Theoretical Biology 235, 275 (2005)
16. Sorrentino, F.: Effects of the network structural properties on its controllability. Chaos 17(3), 033101 (2007)
17. Petermann, T., Rios, P.D.L.: The role of clustering and gridlike ordering in epidemic spreading. Physical Review E 69, 066116 (2004)
18. Grabowski, A., Kosiński, R.A.: Epidemic spreading in a hierarchical social network. Physical Review E 70(3), 031908 (2004)
19. Grabowski, A., Kosiński, R.A.: The SIS Model of Epidemic Spreading in a Hierarchical Social Network. Acta Physica Polonica B 36, 1579 (2005)
20. Verdasca, J., da Gama, M.M.T., Nunes, A., Bernardino, N.R., Pacheco, J.M., Gomes, M.C.: Recurrent epidemics in small world networks. Journal of Theoretical Biology 233(4), 553–561 (2004)
21. Barthélemy, M., Barrat, A., Vespignani, A.: The role of geography and traffic in the structure of complex networks. Advances in Complex Systems (ACS) 10(01), 5–28 (2007)
22. Barrat, A., Barthelemy, M., Pastor-Satorras, R., Vespignani, A.: The architecture of complex weighted networks. Proc. Natl. Acad. Sci. U.S.A. 101(11), 3747–3752 (2003)
23. Restrepo, J.G., Ott, E., Hunt, B.R.: Characterizing the dynamical importance of network nodes and links. Physical Review Letters 97(9), 094102 (2006)
24. Masuda, N., Konno, N.: Multi-state epidemic processes on complex networks. Journal of Theoretical Biology 243, 64 (2006)
25. Aalen, O., Farewell, V., De Angelis, D., Day, N., Gill, O.: New therapy explains the fall in AIDS incidence with a substantial rise in number of persons on treatment expected. AIDS 13(1), 103–108 (1999)
26. Eubank, S., Guclu, H., Anil, M.M.V., Srinivasan, A., Toroczkai, Z., Wang, N.: Modelling disease outbreaks in realistic urban social networks. Nature 429(6988), 180–184 (2004)
27. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C.: Realistic, mathematically tractable graph generation and evolution, using Kronecker multiplication. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS, vol. 3721, pp. 133–145. Springer, Heidelberg (2005)
28. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science 286, 509 (1999)
29. Quax, R.: Modeling and simulating the propagation of infectious diseases using complex networks. Master's thesis, Georgia Institute of Technology (2008)
30. Madduri, K.: A High-Performance Framework for Analyzing Massive Complex Networks. PhD thesis, Georgia Institute of Technology (2008)
31. Saad, Y.: Krylov subspace methods on supercomputers. SIAM J. Sci. Stat. Comput. 10, 1200–1232 (1989)

A Nonlinear Master Equation for a Degenerate Diffusion Model of Biofilm Growth*

Hassan Khassehkhan[1], Thomas Hillen[2], and Hermann J Eberl[1]

[1] Dept. of Mathematics and Statistics,
University of Guelph Guelph, ON, Canada, N1G 2W1
{hkhasseh,heberl}@uoguelph.ca

[2] Dept. of Mathematical and Statistical Sciences, Centre for Mathematical Biology
Univ. of Alberta, Edmonton, AB, Canada, T6G 2G1
thillen@math.ualberta.ca

Abstract. We present a continuous time/discrete space model of biofilm growth, starting from the semi-discrete master equation. The probabilities of biomass movement into neighboring sites depend on the local biomass density and on the biomass density in the target site such that spatial movement only takes place if (i) locally not enough space is available to accommodate newly produced biomass and (ii) the target site has capacity to accommodate new biomass. This mimics the rules employed by Cellular Automata models of biofilms. Grid refinement leads formally to a degenerate parabolic equation. We show that a set of transition rules can be found such that a previously studied ad hoc density-dependent diffusion-reaction model of biofilm formation is approximated well.

1 Introduction

Most bacterial populations live as microbial depositions on immersed surfaces (called substratum in the biofilm context). These biofilm colonies are not a consequence of active or even deliberate aggregation of microorganisms but of immobilization and cell division. Once cells become sessile they start to produce an extracellular polymeric network in which they are themselves embedded (EPS). Thus, they are heavily restricted in their mobility. Living in biofilm colonies is quite different from living as a suspended population. For example, the colony offers protection against mechanical washout or antibiotic attacks.

As long as the conditions are locally favorable to sustain microbial growth, cells will increase in size and eventually divide. The local expansion of a biofilm is primarily driven by this growth process. As new cells require more space, the neighbors must make way. The bacteria closer to the food source grow faster than the bacteria further away, e.g. the ones close to the substratum. This can lead to the characteristic "biofilm mushroom structures" [10] which appear as if the biofilm is growing toward the region with higher food concentrations. In

* Supported by the *Advanced Foods and Materials Network* (AFMNET) [HJE] and the Natural Science and Engineering Research Council of Canada (NSERC) [TH].

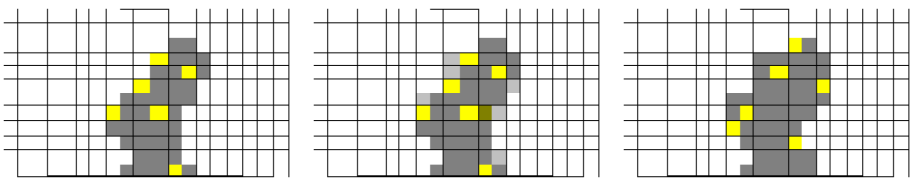


Fig. 1. Schematics of one time step of biomass spreading in cellular automata models, cf [12,13], (left to right): If the biomass in a grid cell approaches or exceeds maximum packing density (yellow grid cells), mass is moved into neighboring sites. If empty neighboring grid cell are found the excess biomass is placed there (grey), otherwise re-shuffling takes place (existing biomass is placed in a neighboring cell [grey-yellow], the excess biomass of which is placed in a suitable empty site).

contrast to chemotaxis, however, this is not an active movement up the nutrient gradient but a consequence of the fact that life under conditions of abundance is more conducive to population growth than under conditions of starvation.

The most widespread technique in modern biofilm modeling, and in fact the modeling concept that first was used to describe multi-dimensional biofilm formation are cellular automata, e.g. [2,7,12,13,14], cf Fig. 1. Deterministic continuum models include the density-dependent diffusion model [3], which is formulated as a traditional spatially structured population and resource dynamics model. Despite the good phenomenological agreement between different model approaches [10,17] no attempts have been made so far to relate these seemingly so different models to each other. We formulate here a discrete space, continuous time model, starting from the master equation that describes the probability that bacteria move from one site on a regular lattice into a neighboring site and *vice versa*. This movement follows stochastic rules which qualitatively mimic those in [12]. The probability of biomass movement into neighboring sites has to account for two particular aspects: (i) It depends on the availability of space in that site (volume filling, cf [11]), and (ii) as long as there is capacity to accommodate new biomass locally the incentive to move into a neighboring site is small (quenching). Semi-discrete master equation models lend themselves in an often straightforward manner to deriving deterministic continuous models. Our goal is to derive from this spatially discrete description the phenomenological nonlinear diffusion model [3]. In fact, since this was originally introduced in an ad hoc fashion, the semi-discrete approach described here can be understood as an a posteriori derivation of this model.

Semi-discrete master equation models for spatially structured populations have been developed for many different types of migration in ecology and cell biology, such as aggregation and (chemo-)taxis, cf. [1,9,11,16]. The advantage of the approach is that phenomenological migration rules are easily formulated based on assumptions on the individual level, while the well-developed machinery of differential equations can be used to study the model on the population level. The biofilm model presented here has qualitative properties that are distinct from other master equation models, since it has to account for the properties (i) and (ii) above. Moreover, since the driving force behind spatial expansion of the biofilm is population growth, this needs to be included in every biofilm model.

2 The Spatially Discrete Master Equation

We consider an equidistant discretization of the real line and denote by $\mathcal{T}_i^{\pm}$ the probability that biomass moves from the ith grid cell into neighbor cell $i \pm 1$, cf. Fig. 2. We follow common practice and equate the dependent variable u_i with the population density in the ith site on the lattice. Like most biofilm models, we subsume EPS and cells in this variable. Since the site's capacity to accommodate bacteria is limited, we normalize the density with respect to the maximum density, i.e. we interpret it as the volume fraction of site i occupied by the population. Thus, $0 \leq u_i \leq 1$ is mandatory. The $\mathcal{T}_i^{\pm}$ can also be understood as mass transfer rates. The master equation reads

$$\frac{du_i}{dt} = \mathcal{T}_{i-1}^{+} u_{i-1} + \mathcal{T}_{i+1}^{-} u_{i+1} - (\mathcal{T}_i^{+} + \mathcal{T}_i^{-}) u_i + \mathcal{K}_i u_i, \tag{1}$$

where $\mathcal{T}_i^{\pm} = \alpha q_i^{\pm}$ with a scaling factor α that depends on time-scale and length scale, i.e. distance between two sites h. More specifically, for diffusion problems α scales with h^2 , such that $\lim_{h \to 0} \alpha h^2 = \alpha_0 > 0$. The probability $q_i^{\pm}$ of a jump from location i to location $i \pm 1$ depends on the densities in both sites, i.e. on u_i and on $u_{i\pm1}$. We make the general *ansatz*

$$q_i^{\pm} = q(u_i) p(u_{i\pm1}). \tag{2}$$

In cellular automaton biofilm models like [12], spatial spreading of biomass takes place if the local biomass density u_i reaches or exceeds the maximum cell packing density, i.e. $u \approx 1$. Then a given or randomly chosen amount of the local biomass density is placed in a neighboring grid cell according to some stochastic local rule. In the continuum model [3], on the other hand, the spatial spreading is described deterministically: biomass is not moved into neighboring sites as long as newly produced biomass still easily fits into the local site. As u_i increases and gets close to 1, $\mathcal{T}_i^{\pm}$ increases. Moreover, the bigger the volume fraction in the target site, the smaller is $\mathcal{T}_i^{\pm}$. Thus, we assume $q(u)$ to be a monotonously increasing function and $p(u)$ to be a monotonously decreasing function. Assuming sufficient smoothness, we have for p, q the properties

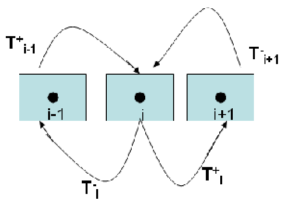


Fig. 2. Schematic of movement of biomass between neighboring lattice cells. The probability for biomass to move from cell i into cell $i \pm 1$ is denoted by $\mathcal{T}_i^{\pm}$ etc.

$$\begin{aligned} q(0) = 0, \quad q'(u) \geq 0, \quad q(1) = 1, \\ p(0) = 1, \quad p'(u) \leq 0, \quad p(1) = 0. \end{aligned} \tag{3}$$

In (1), $\mathcal{K}_i$ is the net biomass production rate in grid cell i. This is the actual force driving the expansion of biofilm colonies. If $\mathcal{K}_i$ is a positive constant the model describes unrestricted exponential growth. In most biofilm systems, growth is eventually not everywhere unlimited. Limitations can be induced by local limitation of nutrients, iron, oxygen (aerobic case) etc. or by large amounts of growth inhibitors such as protonated lactic acids, proton ions [i.e. low pH], or oxygen (anaerobic case). Modeling these effects is conceptually straightforward. Each growth limiting substrate is described by an additional transport-reaction equation, which is coupled with (1) in the reaction term $\mathcal{K}_i$. The growth limitation mechanisms listed above are due to an increase of biomass in the system. For example, the more bacteria there are in the system the more nutrients are depleted, etc. In many biofilm systems the net growth rate can become negative, e.g if nutrients are locally completely depleted and natural cell loss dominates. Under this light, assuming $\mathcal{K}_i = const$ is the most challenging case because biomass production is highest and, thus, biomass spreading most pronounced.

The relationship between the spatially discrete and a continuous description is established by passing the grid cell size h in (1) to the continuous limit, $h \to 0$. Assuming enough smoothness we approximate $q(u_{i\pm1})$ and $p(u_{i\pm1})$ in the usual way by Taylor polynomials about u_i,

$$\begin{aligned} q(u_{i\pm1}) = q(u_i) + (u_{i\pm1} - u_i)q'(u_i) + (u_{i\pm1} - u_i)^2 q''(u_i)/2 + \mathcal{O}((u_{i\pm1} - u_i)^3) \\ p(u_{i\pm1}) = p(u_i) + (u_{i\pm1} - u_i)p'(u_i) + (u_{i\pm1} - u_i)^2 p''(u_i)/2 + \mathcal{O}((u_{i\pm1} - u_i)^3). \end{aligned}$$

Interpreting u_i as a quantity in the cell center x_i, we can interpolate the grid function u_i by a continuous function u with $u(t, x_i) = u_i(t)$. For given t we approximate $u(t, x_{i\pm1})$ by Taylor polynomials

$$u(t, x_{i\pm1}) = u_i(t) \pm h\frac{\partial u_i(t)}{\partial x} + \frac{h^2}{2}\frac{\partial^2 u_i(t)}{\partial x^2} + \mathcal{O}(h^3)$$

where $\frac{\partial u_i}{\partial x}$ is short-hand for $\frac{\partial u}{\partial x}(\cdot, x_i)$. Substituting these expressions into (1) we obtain after dropping $\mathcal{O}(h^3)$ terms and sorting by powers of $\partial u_i / \partial x$

$$\begin{aligned} \frac{\partial u_i}{\partial t} = \mathcal{K}u_i + \alpha h^2 \left[p(u_i)q(u_i) + u\left(p(u_i)q'(u_i) - q(u_i)p'(u_i)\right)\right] \frac{\partial^2 u_i}{\partial x^2} + \\ + \alpha h^2 \left[p(u_i)q''(u_i)u_i + 2q'(u_i)p(u_i) - u_i q(u_i)p''(u_i)\right] \left(\frac{\partial u_i}{\partial x}\right)^2. \end{aligned} \tag{4}$$

Defining now the density dependent diffusion coefficient

$$D(u) := p(u)q(u) + u\left(p(u)q'(u) - q(u)p'(u)\right) \tag{5}$$

equation (4) can be written as a diffusion-reaction equation in divergence form

$$\frac{du}{dt} = \frac{\partial}{\partial x}\left(D(u)\frac{\partial u}{\partial x}\right) + \mathcal{K}u. \tag{6}$$

3 An Inverse Problem

A density-dependent diffusion-reaction model for biofilm formation was introduced in [3] and has been studied since numerically and analytically in its original and extended versions, e.g. in [4,5,6,8]. The pecularity of that model lies in the form of the nonlinear diffusion coefficient $D(u)$, namely

$$D(u) = \delta u^a (1-u)^{-b}, \quad a, b \geq 1. \tag{7}$$

For small $u \ll 1$ this type of nonlinear diffusion acts like the porous medium equation, while for large $u \approx 1$ it shows fast diffusion effects. The interplay of both effects is required to describe a growing biofilm. It was shown in [6] that solutions of this model remain bounded by and separated from unity, $u < 1 - \eta$, for a $\eta > 0$, as long as the biofilm/water interface stays away from the boundary of the domain somewhere. Thus, the singularity is never reached. It follows then with standard arguments about degenerate diffusion-reaction equations, cf [15], that initial data with compact support imply solutions with compact support. In our application this means that the interface between biofilm and surrounding liquid is sharp and propagates at finite speed.

We try to find jump probabilities $q(u)$ and $p(u)$ with (3) such that the diffusion-reaction model (6) with (7) is recovered. This is a constrained scalar boundary value problem for two unknown functions. It is transformed into an ordinary boundary value problem by introducing one more assumption on q and/or p. We propose

$$p(u) := 1 - uq(u). \tag{8}$$

It is easily verified that p satisfies (3) if q does and *vice versa*. With (8) equation (5) becomes the linear ordinary differential equation

$$uq'(u) + q(u) = D(u), \tag{9}$$

which degenerates for $u = 0$. Moreover, the right hand side of (9) blows up for $u = 1$. We introduce the following regularization with small parameter $\epsilon > 0$, which was already used in the analysis of the biofilm model (6), (7) in [6,8].

$$D_\epsilon(u) = \begin{cases} \delta \frac{(u+\epsilon)^a}{(1-u)^b}, & \text{for} \quad u < 1 - \epsilon, \\ \delta \epsilon^{-b}, & \text{for} \quad u \geq 1 - \epsilon. \end{cases} \tag{10}$$

The unique solution q_ϵ of the regularized initial value problem

$$(u + \epsilon) q'(u) + q(u) = D_\epsilon(u), \quad q(0) = 0 \tag{11}$$

is obtained as the strictly increasing function

$$q_\epsilon(u) = \frac{1}{u + \epsilon} \int_0^u D_\epsilon(s) ds,$$

which is bounded for every $\epsilon > 0$ and $0 \leq u \leq 1$. Moreover, $1 < q_\epsilon(1) < \infty$ for every small enough ϵ. Thus, continuity and monotonicity imply that there exists a $\tilde{u}_\epsilon < 1$ such that $q_\epsilon(\tilde{u}_\epsilon) = 1$, while $q_\epsilon(u) < 1$ for $u < \tilde{u}_\epsilon$ and $q_\epsilon(u) > 1$ for $u > \tilde{u}_\epsilon$. Hence, $q_\epsilon(u)$ does not belong to the desired class of functions

$$\mathcal{G} := \{g \in \mathcal{C}([0,1]) : g(0) = 0, \quad g(1) = 1, \quad g(u_1) \le g(u_2) \quad \forall\, u_1 < u_2\}.$$

Thus the problem of finding an exact solution of (11) in $\mathcal{G}$ is ill-posed and the best we can hope for is to find a function $r_\epsilon(u) \in \mathcal{G}$ that solves (11) approximately. Thus, we are looking for the function in $\mathcal{G}$ with the smallest distance from q_ϵ,

$$\min_{r \in \mathcal{G}} \|q_\epsilon - r\|_2. \tag{12}$$

This minimization problem is solved by the function

$$r_\epsilon(u) = \begin{cases} q_\epsilon(u), & \text{for} \quad u \le \tilde{u}_\epsilon, \\ 1, & \text{for} \quad u > \tilde{u}_\epsilon, \end{cases} \tag{13}$$

because $r_\epsilon \in \mathcal{G}$ and

$$\|q_\epsilon - r_\epsilon\|_2^2 = \int_0^{\tilde{u}_\epsilon} (q_\epsilon(s) - r_\epsilon(s))^2 ds + \int_{\tilde{u}_\epsilon}^1 (q_\epsilon(s) - r_\epsilon(s))^2 ds.$$

The first integral vanishes for $r(u) = r_\epsilon(u)$ as defined above, and, since $q_\epsilon(u) > 1$ and $r(u) \le 1$ for $u > \tilde{u}_\epsilon$, the second integral becomes minimal if $r(u) = 1$ for $u > \tilde{u}_\epsilon$. Passing the regularization parameter to the limit, $\epsilon \to 0$, then $r_\epsilon(u) \to r(u)$ pointwise, where

$$r(u) = \begin{cases} \phi(u), & \text{for} \quad u \le \tilde{u}, \\ 1, & \text{for} \quad u > \tilde{u}, \end{cases}, \quad \text{with} \quad \phi(u) = \frac{1}{u}\int_0^u D(s)ds. \tag{14}$$

Function $\phi(u)$ is the strictly increasing solution of the initial value problem (9) with $q(0) = 0$ and $\tilde{u}$ is the unique value such that $\phi(\tilde{u}) = 1$. Note that $\lim_{u\to 1}\phi(u) = \infty$. Obviously $r \in \mathcal{G}$. We choose $q(u) := r(u)$. It remains to validate how good this approximation is for practical purposes. We recall from [6] that the solution of (6) with (7) satisfies $u < 1-\eta$ for a $\eta > 0$, which depends on the parameters and the initial data. Thus, if $\tilde{u} > 1-\eta$ then (6) with (7) is equivalent with the PDE model that one obtains from the lattice model with jump probability $q(u) = r(u)$ from (14). Otherwise they are equivalent as long as the solutions remain bounded by $\tilde{u}$, i.e. for a finite time interval. Unfortunately, the proof in [6] does not allow to compute quantitative estimates for η, so that we have to fall back on comparisons with computer simulations. For example, in the 2D and 3D simulations of the single-species growth model in [4,5,6], the solution reached values above $u \approx 0.99$ but remained below $u \approx 0.9999$, thus, we have the numerical estimates $\hat{\eta} > 10^{-4}$. In all three cases the exponents in the biomass diffusion coefficient were $a = b = 4$ and the biofilm motility coefficient $\delta < 10^{-12}$. Then

$$\int_0^u D(s)ds = u\phi(u) = \delta\left(u + 4\ln(1-u) - \frac{-18u^2 + 30u - 13}{3(1-u)^3} - \frac{13}{3}\right). \tag{15}$$

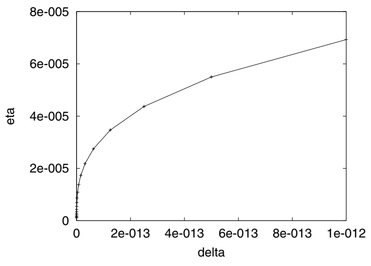


Fig. 3. $\tilde{\eta} = 1 - \tilde{u}$ as a function of the biomass mobility parameter δ

On the other hand the value $\tilde{\eta} := 1-\tilde{u}$ can be computed from (15). It is plotted in Fig. 3 for δ covering several orders of magnitude. $\tilde{\eta}$ decreases as δ decreases. In all cases, $\tilde{u}$ remained above the maximum values of u in the numerical simulations reported above, i.e $\hat{\eta} > 1 - \tilde{u} =: \tilde{\eta}$ by at least one order of magnitude. Thus the nonlinear diffusion-reaction model that is derived from the semi-discrete master equation (1) with the jump probabilities $T_i^{\pm}(u) = \alpha q(u_i)p(u_{i\pm1})$, $q(u) := r(u)$, $p(u) := 1 - uq(u)$ is in these simulations indeed equivalent to the double-degenerate density-dependent diffusion-reaction model.

We remark that (6) can also be written in the Laplacian form

$$u_t = \frac{\partial^2}{\partial x^2}\left(\int_0^u D(s)ds\right) + \mathcal{K}u. \tag{16}$$

Hence,(1) is equivalent with the finite difference discretisation of (16)

$$\frac{du_i}{dt} = \alpha\left(u_{i-1}q(u_{i-1}) - 2u_i q(u_i) + u_{i+1}q(u_{i+1})\right) + \mathcal{K}u_i. \tag{17}$$

Note that assuming $p(u) \equiv 1$ instead of (8), i.e. transition probability does not depend on density in target site, also leads to (9) and thus to (17).

4 Numerical Results

We conduct computer simulations of the semi-discrete model in two space dimensions on a $n \times m$ grid. Nutrients are assumed to be nowhere limited, $\mathcal{K} = const.$ In interior grid points $1 < i < n$, $1 < j < m$ the 2D variant of (1) reads

$$\begin{aligned}\frac{d}{dt}u_{i,j} = \mathcal{K}u_{i,j} + \mathcal{T}_{i-1,j}^{+,0}u_{i-1,j} + \mathcal{T}_{i+1,j}^{-,0}u_{i+1,j}\mathcal{T}_{i,j-1}^{0,+}u_{i,j-1} + \mathcal{T}_{i,j+1}^{0,-}u_{i,j+1} \\ - (\mathcal{T}_{i,j}^{+,0} + \mathcal{T}_{i,j}^{-,0} + \mathcal{T}_{i,j}^{0,+} + \mathcal{T}_{i,j}^{0,-})u_{i,j},\end{aligned} \tag{18}$$

where $\mathcal{T}_{i,j}^{\pm,0} = \alpha q(u_{i,j})p(u_{i\pm1,j})$ and $\mathcal{T}_{i,j}^{0,\pm} = \alpha q(u_{i,j})p(u_{i,j\pm1})$. The derivation of the 2D equivalents of (4) and (6) follows the same procedure as in 1D, albeit notation and calculations are more cumbersome. The inverse problem in Section 3 is independent of the problem dimension.

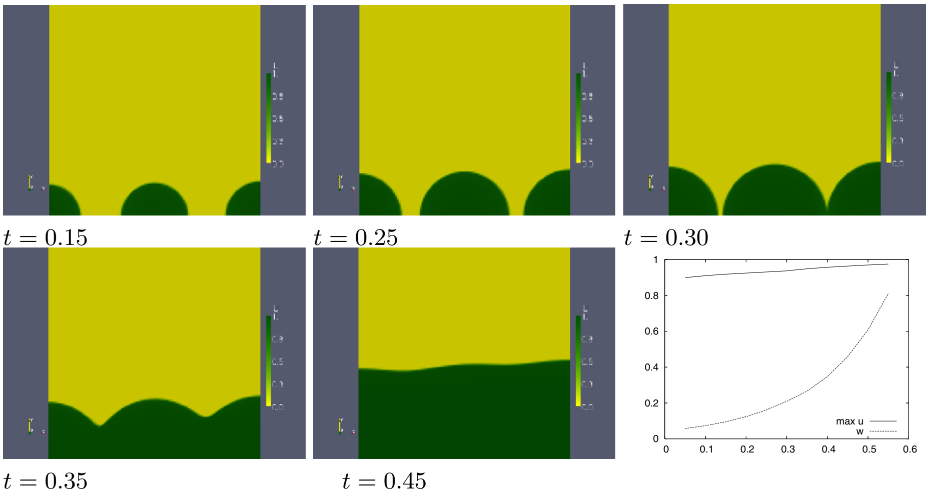


Fig. 4. 2D Simulation of biofilm growth on a 128 × 128 grid. Shown are for selected time instances t the biomass densities $u_{i,j}$.The bottom right insert shows $\max_{ij} u_{ij}(t)$ and the occupancy function $\omega(t)$, i.e. the fraction of grid cells with $u > 10^{-6}$.

In $i = 1$, $i = n$, $j = 1$, $j = m$, (18) is corrected such that no exchange of biomass across the lattice boundaries takes place, mimicking homogeneous Neumann conditions. We set in the "virtual grid points" $u_{0,j} := u_{1,j}$, $u_{n+1,j} := u_{n,j}$, $u_{i,0} := u_{i,1}$, $u_{i,m+1} := u_{i,m}$ and eliminate them from (18). The system of nm ordinary differential equations is integrated by the Runge-Kutta-Fehlberg method RKF4(5).

We use a 128×128 grid. Cells are inoculated in three spherical pockets around the grid cells $(i, j) = (1, 1)$ [radius $\rho = 0.1n$, initial biomass density $u_0 = 0.8$], $(i, j) = (n/2, 1)$ [$\rho = 0.11n$, $u_0 = 0.75$], and $(i, j) = (n, 1)$ [$\rho = 0.12n, u_0 = 0.7$]. We take $q(u)$ from (14) and $p(u) = 1 - uq(u)$. For the coefficients of $D(u)$ we pick $a = b = 4$ and $\delta = 10^{-12}m^2/d$, for the biomass production rate $\mathcal{K} = 6/d$. With a simulation domain of $100 \times 100\mu m^2$, h corresponds to $0.78125\mu m$.

In Fig. 4 we show for selected t the biomass density $u_{i,j}(t)$, interpolated between grid cells by the visualization software. Initially the three colonies

Table 1. Results of a grid refinement study. Reported are the values $N_1 := \frac{1}{nm}\sum_{i,j} |u_{ij}(t_p) - \tilde{u}_{ij}(t_p)|$ for three selected time steps, where $\tilde{u}$ is the projection of the coarser grid onto the finer grid.

grid refinement	$t = 0.15$	$t = 0.25$	$t = 0.30$
$h = 1/16 \to 1/32$	0.025910	0.046667	0.056930
$h = 1/32 \to 1/64$	0.010767	0.017459	0.021014
$h = 1/64 \to 1/128$	0.003537	0.005473	0.006467
$h = 1/128 \to 1/256$	0.001828	0.002741	0.003442
$h = 1/256 \to 1/512$	0.000873	0.001184	0.001365

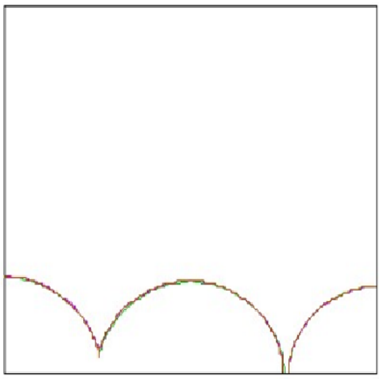


Fig. 5. Biofilm/water interface at $t = 30$ for grid sizes $h = 1/128, 1/256, 1/512$ [green, magenta, brown]

solidify, i.e. u increases, first without notable expansion. Later, the colonies expand spherically. Eventually, neighboring colonies merge and a homogeneous thick layer of biomass develops that expands exponentially. This is a consequence of the abundance of food and agrees with all other biofilm models. The simulation is stopped before the entire lattice is filled with biomass. The biomass density remains clearly below the cut-off value $\tilde{u}$ that was computed above.

In order to investigate how the solution of the spatially discrete model changes with grid refinement, we repeat and compare the simulations on grids of size $2^k \times 2^k$, where $k = 4, ..., 9$. Table 1 shows the convergence of the grid refinement study. This is visually supported by Fig. 5 where the biofilm/water interface is plotted for subsequent grids. These agree well within plotting accuracy.

5 Conclusion, Discussion, and Future Work

Spatially discrete master equations are routinely used to derive population models that can be studied with the well developed machinery of partial differential equations. We use this approach to derive a highly nonlinear diffusion-reaction model of biofilm expansion. The underlying mechanisms are quite different than for other biological systems. Most notably, spatial spreading of biomass is driven by production of new biomass and takes place only if locally no space is available to accommodate newly produced cells. In the continuous limit this is rendered by a combination of degenerate and fast diffusion. In numerical simulations we could show that the spatially discrete master equation and the continuous model agree well. The assumptions that we made to define the probabilities for biomass movement mimic the stochastic rules that are used in cellular automata biofilm models. Therefore, we understand the master equation model as a link between discrete stochastic and continuous deterministic biofilm models.

In our current study the equivalence of spatially discrete and continuous model rests on empirical computer simulations. While it follows from continuity that both models agree exactly for some time, the question how long this equivalence is valid certainly warrants more rigorous mathematical analysis. On the modeling side, future research will involve the extension of the approach to mixed-culture biofilm systems and to biofilm systems with preferred spreading directions, e.g. biofilms that predominantly creep over the substratum rather than forming patchy biofilm colonies. Moreover, for realistic applications the biofilm population models need to be coupled with models of resource dynamics, as already indicated above.

References

1. Anguige, K., Schmeiser, C.: A one-dimensional model of cell diffusion and aggregation, incorporating volume filling and cell-to-cell adhesion. J. Math. Biol. (in press)
2. Chambless, J.D., Hunt, S.M., Stewart, P.S.: A three-dimensional computer model of four hypothetical mechanisms protecting biofilms from antimicrobials. Appl. Env. Microbiol. 72(3), 2005–2013 (2006)
3. Eberl, H.J., Parker, D.F., van Loosdrecht, M.C.M.: A new deterministic spatio-temporal continuum model for biofilm development. J. Theor. Medicine 3(3), 161–175 (2001)
4. Eberl, H.J., Demaret, L.: A finite difference scheme for a doubly degenerate diffusion-reaction equation arising in microbial ecology. El. J. Diff. Equ. CS 15, 77–95 (2007)
5. Eberl, H.J., Sudarsan, R.: Exposure of biofilms to slow flow fields: the convective contribution to growth and disinfections. J. Theor. Biol. 253(4), 788–807 (2008)
6. Efendiev, M.A., Zelik, S.V., Eberl, H.: Existence and longtime behavior of a biofilm model. Comm. Pure Appl. Analysis 8(2), 509–531 (2009)
7. Hermanowicz, S.W.: A simple 2D biofilm model yields a variety of morphological features. Math. Biosc. 169(1), 1–14 (2001)
8. Khassehkhan, H., Efendiev, M.A., Eberl, H.J.: A degenerate diffusion-reaction model of an amensalistic probiotic biofilm control system: Existence and simulation of solutions. Discr. Cont. Dyn. Sys. B (in press)
9. Lizana, M., Padorn, V.: A spatially discrete model of aggregating populations. J. Math. Biol. 38, 79–102 (1999)
10. Van Loosdrecht, M.C.M., Heijnen, J.J., Eberl, H.J., Kreft, J.U., Picioreanu, C.: Mathematical modelling of biofilm structures. Ant. v. Leeuwenhoek 81, 245–256 (2002)
11. Painter, K., Hillen, T.: Volume-Filling and Quorum Sensing in Models for Chemosensitive Movement. Can. Appl. Math. Quart. 10(4), 501–543 (2002)
12. Picioreanu, C., van Loosdrecht, M.C.M., Heijnen, J.J.: A new combined differential-discrete cellular automaton approach for biofilm modeling: Application for growth in gel beads. Biotech. Bioeng. 57(6), 718–731 (1998)
13. Pizarro, G., Griffeath, D., Noguera, D.: Quantitative cellular automaton model for biofilms. J. Env. Eng. 127(9), 782–789 (2001)
14. Pizaro, G.E., Texiera, J., Sepulveda, M., Noguera, D.: Bitwise implementation of a 2D cellular automata biofilm model. J. Comp. Civ. Eng. 19(3), 258–268 (2005)
15. Samarskii, A.A., Galaktionov, V.A., Kurdyumov, S.P., Mikhailov, A.P.: Blow-up in quasilienar parabolic equations. de Gruyter, Berlin (1995)
16. Visser, A.W.: Lagrangian modelling of plankton motion: From deceptively simple random walks to Fokker–Planck and back again. J. Mar. Sys. 70(3-4), 287–299 (2008)
17. Wanner, O., Eberl, H., Morgenorth, O., Noguera, D., Picioreanu, D., Rittmann, B., van Loosdrecht, M.: Mathematical Modelling of Biofilms. IWAPublishing, London (2006)

Graphical Notation for Diagramming Coupled Systems

J. Walter Larson[1,2,3]

[1] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA
larson@mcs.anl.gov
[2] Computation Institute, University of Chicago, Chicago, IL USA
[3] Department of Computer Science, The Australian National University
Canberra ACT 0200 Australia

Abstract. Multiphysics and multiscale–or *coupled*–systems share one fundamental requirement: Construction of coupling mechanisms to implement complex data exchanges between a system's constituent models. I have created a graphical schema for describing coupling workflows that is based on a theoretical framework for describing coupled systems. The schema combines an expanded set of traditional flowchart symbols with pictograms representing data states. The data pictograms include distributed mesh, field, and domain decomposition descriptors and spatiotemporal integration and accumulation registers. Communications pictograms include: blocking- and non-blocking point-to-point and $M \times N$ parallel data transfer; parallel data transposes; collective broadcast, scatter, gather, reduction and barrier operators. The transformation pictograms include: intergrid interpolation; spatiotemporal integral operators for accumulation of state and flux data; and weighted merging of output data from multiple source models for input to a destination model. I apply the schema to simple problems illustrating real situations in coupler design and implementation.

1 Introduction

Coupled systems are increasingly prevalent in computational science and engineering. Multiphysics models combine subsystem models to achieve higher-fidelity simulation of the greater whole and superior solutions in their constituent processes. Multiscale models capture spatiotemporal-scale interactions by coupling separate models that operate on disparate time and length scales. Multiphysics and multiscale models share a common requirement for data exchange mechanisms that allow their constituent subsystems to compute their respective states—the *coupling problem* (CP) [1]. In many cases, coupled systems contain constituent subsystems possessing high levels of computational complexity that warrant parallel processing. Coupling in distributed-memory parallel environments is particularly difficult—the *parallel coupling problem* (PCP) [1].

Most coupled models are built by multidisciplinary teams from legacy model codes. Coupling mechanisms—or *couplers*—confront coupled model designers with complexity in choices of operation order (e.g., computation of fluxes followed by interpolation, or vice versa) and algorithms (e.g., $M \times N$ data transfer), creating the potential for uncertainty or even software bugs. This raises the question: Is there a compact way to express coupling workflow complexity that allows rapid, up-front analysis of coupler design? In this paper I propose a graphical schema for elucidating coupling workflows and employ it to explore coupler design.

Symbols and symbolic diagrams are widely used in place of words to communicate ideas. Symbol sets for diagrams are available for various disciplines, including electrical engineering, meteorology, and computer science [2]. Computational workflows traditionally are characterized by using flowcharts [3,4] that specify control flow and processing operations in a system. Flowcharts, however, depict processing at a high level of granularity—in some cases line by line of code. If we are to use flowcharts, it must be at a coarse level of code granularity, but with sufficient detail to capture often-repeated operations such as intergrid interpolation. Data flow diagrams [5] capture the data states in various parts of a system and how information flows through a system; they do not offer detail pertaining to the transformations driving the data flows. The Unified Modeling Language (UML) [6] provides diagrams for elucidating system structure and behavior. The activity diagram depicts a series of processing steps in a system; ovals represent processing activities and their progression is represented by connecting arrows. The state diagram depicts a series of states of a system; squashed boxes represent system states, connected by labeled arrows denoting processing steps or guard expressions. UML's graphics are easy to draw, but subtle, and it's easy for nonspecialists such as computational scientists to make errors using UML [7]. None of these existing solutions suffices to cover the complete problem of documenting the transformations and data states of a coupling workflow.

I have concluded that an approach that incorporates elements of flowcharts and data flow diagrams will best suit the problem at hand. My graphical schema leverages flowcharts but augments some of its well-known symbols with pictograms that represent processing activities relevant to coupling. The schema is derived from a theoretical framework for the CP and PCP. In Section 2 I summarize this theoretical framework, defining terms for both the schema and discussion for the remainder of this paper. In Section 3 I define the schema's pictograms and drawing conventions. In Section 4 I construct schematics illustrating some commonly encountered coupling mechanisms.

2 Coupling in Multiphysics and Multiscale Models

Below I define terms and provide a theoretical overview of data trafficking in the CP and PCP; further details are available in [1]. A coupled system is constructed from N interacting models—or *constituents*—$\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$. Each model has a spatial domain Γ_i plus time; intersections between spatial domains result in *overlap domains* $\Omega_{ij} = \Gamma_i \cap \Gamma_j$. Each model's domain boundary $\partial\Gamma_i$ is the intersection of its overlap domains; that is, $\partial\Gamma_i \equiv \cap_{j \neq i} \Omega_{ij}$. Coupling entails data exchange between models. Denote each model's state variables, inputs, and outputs as (U_i, V_i, W_i). Each of these entities is a set of variables; for example, U comprises the wind, temperature, pressure, and humidity fields for a simple atmosphere model. A model solves its equations of evolution on a spatial domain Γ; thus the state on the domain is a vector field resulting from the Cartesian product $U \times \Gamma$. A model's inputs (outputs) is also a vector field $V \times \partial\Gamma$ $(W \times \partial\Gamma)$.

Models $\mathcal{C}_i$ and $\mathcal{C}_j$ are *coupled* if and only if at a minimum $\mathcal{C}_i$ provides output to (receives input from) $\mathcal{C}_j$. This requires $\Omega_{ij} \neq \emptyset$. In some cases the data dependency relationship is immediately obvious; that is, $W_i \cap V_j \neq \emptyset$ or $W_j \cap V_i \neq \emptyset$. In other cases, V_i (V_j) is computable from W_j (W_i) by a *coupling transformation* $\mathscr{T}_{ij} : W_j \rightarrow V_i$ $(\mathscr{T}_{ji} : W_i \rightarrow V_j)$. Thus far we have described *explicit coupling. Implicit coupling* between models $\mathcal{C}_i$ and $\mathcal{C}_j$ constitutes an overlap between their respective state variables; that is, $U_i \cap U_j \neq \emptyset$. Implicit coupling requires a self-consistent, simultaneous solution for shared-state variables using a *solver* $\mathscr{S}_{ij}$ (ordering of indices i and j is irrelevant).

Coupling events play a crucial role in the time evolution of multiscale and multiphysics systems. The time signature of the data exchanged is either *instantaneous* or *integrated*; integrated data exchanges involve the delivery of time integrated (averaged) flux (state) data from a source model to a destination model, which applies integrated fluxes incrementally during intervals between coupling events. Some coupled systems employ integrated data delivery to loosen intermodel couplings; see Section 4 of [1] for further discussion. Similarly, multiscale models may use spatially integrated data delivery to transfer information from smaller to larger length scales.

Thus far we have discussed bipartite coupling. In principle, a constituent $\mathcal{C}_i$ can receive the same input data from more than one model—for example, $\mathcal{C}_j$ and $\mathcal{C}_k$. In this situation, *merging* of data is required if there is a second-order overlap domain $\Omega_{ijk} \equiv \Gamma_i \cap \Gamma_j \cap \Gamma_k \neq \emptyset$ and $\mathscr{T}_{ij}$ and $\mathscr{T}_{ik}$ produce some of the same input fields among V_i. Higher-order merges may occur on higher-order overlap domains, for example, a $k-1$-way merge on the kth order overlap domain $\Omega_{n_1,\ldots,n_k} \equiv \Omega_{n_1} \cap \cdots \cap \Omega_{n_k}$ for $n_1 \neq \cdots \neq n_k$. Multipartite (k-way) implicit coupling occurs on $\Omega_{n_1,\ldots,n_k}$ if $U_{n_1} \cap \cdots \cap U_k \neq \emptyset$ for $n_1 \neq \cdots \neq n_k$; in this case a k-way self-consistent solver $\mathscr{S}_{n_1,\ldots,n_k}$ is required.

Coupled system models are implemented on digital computers by using numerical analysis techniques that discretize space and time. A model's gridpoints derive from a *discretization* $\mathbf{\Delta}_i(\Gamma_i)$; its boundary gridpoints are $\mathbf{\Delta}_i(\partial\Gamma_i)$. The state, inputs, and outputs of a numerical model $\mathcal{C}_i$ are its *state vector* $\mathbf{U}_i \equiv U_i \times \mathbf{\Delta}_i(\Gamma_i)$, *input vector*, $\mathbf{V}_i \equiv V_i \times \mathbf{\Delta}_i(\Gamma_i)$, and *output vector* $\mathbf{W}_i \equiv W_i \times \mathbf{\Delta}_i(\Gamma_i)$. Thus an explicit coupling transformation delivering data from $\mathcal{C}_j$ to $\mathcal{C}_i$ is $\mathscr{T}_{ij} : \mathbf{W}_j \rightarrow \mathbf{V}_i$; $\mathscr{T}_{ij}$ will likely comprise a *field variable transformation* $\mathscr{F}_{ij}$ that embodies natural-law relationships between W_j and V_i and a *mesh transformation* $\mathscr{G}_{ij}$ that maps the same variable defined on gridpoints in $\mathbf{\Delta}_j(\Omega_{ij})$ to values defined on the gridpoints in $\mathbf{\Delta}_i(\Omega_{ij})$. In principle, these operations do not commute; that is, $\mathscr{G}_{ij} \circ \mathscr{F}_{ij} \neq \mathscr{F}_{ij} \circ \mathscr{G}_{ij}$. This feature is a source of coupling uncertainty and a motivator for graphical representation of coupling workflows.

Thus far we have assumed a von Neumann uniprocessor computer architecture. Introduction of concurrency complicates the CP, leading to the PCP. In a single address space, the complication is parallelization of the data transformations $\mathscr{T}_{ij}$ and any solvers required by implicit couplings. Distributed-memory architectures introduce further complications—domain decomposition of coupling data, parallel data transfer, and concurrency in model execution. Domain decomposition of data across a pool of K processes is accomplished by the *partitioning operator* $\mathbf{P}(\cdot)$; each model will have its own domain decomposition $\mathbf{P}_i(\cdot)$. Thus, for model $\mathcal{C}_i$ resident on a pool of K_i processes, $\mathbf{P}_i(\Gamma_i) = \{\gamma_i^1, \dots \gamma_i^{K_i}\}$, with γ_i^ν the portion of Γ_i resident on the νth process; similarly, $\mathbf{P}_i(\Omega_{ij}) = \{\omega_{ij}^1, \dots \omega_{ij}^{K_i}\}$, $\mathbf{P}_i(\mathbf{U}_i) = \{u_i^1, \dots, u_i^{K_i}\}$, $\mathbf{P}_i(\mathbf{V}_i) = \{v_i^1, \dots, v_i^{K_i}\}$, and $\mathbf{P}_i(\mathbf{W}_i) = \{w_i^1, \dots, w_i^{K_i}\}$. Parallelization of $\mathscr{T}_{ij}$ and any implicit solvers requires parallel data transfer to deliver data from the source model $\mathcal{C}_j$ to the destination model $\mathcal{C}_i$; this amounts to adding a *data movement operation* $\mathscr{H}_{ij}$ to the mesh and field variable transformations. The data mover $\mathscr{H}_{ij}$ is a one-way parallel data (i.e., $M \times N$) transfer or a two-way parallel data redistribution (i.e., a transpose). *Process composition* is the mapping the models to pools of processes or *cohorts*. On a uniprocessor system, *serial composition* is the only option; models run in turn successively on the single resource. Multiprocessors allow further mapping strategies: *parallel composition*, in which models are mapped to nonoverlapping process pools and execute simultaneously on their respective cohorts; *hybrid composition*, which nests serial and parallel compositions to create complex process maps; and *overlapping composition* in which cohorts of two models *partially* intersect.

3 Graphical Schema Specification

The schema must enable users to capture the functionality present in coupling workflows. It must make clear, at a glance, design choices in terms of parallelism, process composition, and number of executable images. The pictograms must be easy to draw by hand to encourage use on paper and whiteboards. The symbol set must be sufficiently complete to cover a wide variety of coupling functions as outlined in Section 2. The schema must be extensible to allow users to create new symbols for coupling functions specific to their applications. The standard color scheme should be black and white to allow ease of sketching and to allow user-defined color coding as an additional degree of freedom in diagram construction.

Figure 1 displays the symbols and line conventions in the schema. Rectangles with rounded corners represent subsystem models. Model elements are drawn with either solid or dashed lines depending on the model's layout in the system's process composition. Ellipses represent the field, mesh, and domain decomposition data exchanged and processed during coupling. The graphical schema inherits a number of conventions from flowcharting: rectangles with sharp corners represent processing operations; a rhombus indicates a decision point; parallelograms indicate i/o operations; a triangle represents extraction of a subset of data from a data object; an upside-down triangle represents merging of multiple data objects into a single object; and a circle (upside-down house) represents a continuation point to a corresponding continuation point elsewhere on the same diagram (on another diagram). A burst symbol indicates parallel communications such as MPI point-to-point and collective operations. Directed arrows represent processing paths. Processing paths do not cross but may touch in the case of iterative processing. When space is tight in a diagram, one can draw processing paths *appearing* to cross (e.g., Figure 5), but as with flowcharts, the crossing is not significant. One may draw one of the paths with a "bump" in it to represent this skewness—a convention adopted from circuit diagrams. For parallel systems, dashed and dotted lines are employed as needed to identify processing occurring on a subset of the parent cohort; for example, differing types of lines emerging from the decision rhombus with the criterion "MyID == Root?" distinguish processing paths for root and nonroot processes.

The semantics of the schema are defined as follows. A line connecting a data (processing) element to a processing (data) element signifies an input (output) relationship to (from) the processing element. A line connecting two processing elements signifies flow of control in the direction dictated by the arrow; input data to the destination processing element is treated separately. A data element may serve as input to multiple processing elements and a processing element may have multiple outputs.

Model boxes can be annotated with pertinent information such the name of the model, as parallel or uniprocessor, parallelism mechanism, and number of processing elements. Data symbols can be annotated with descriptive information such as model name, grid name, and list of fields. Processing boxes can be annotated with the name of the algorithm embodied by the box; this is also the convention for adding new processing symbols. Continuation symbols are by definition annotated by a label indicating the continuation point; shading or blackening a continuation symbol indicates that the rest of the system in that direction of the workflow is regarded as a black box.

Multiple model boxes with dashed edges represent a serial composition (Figure 2(a)). Multiple model boxes with solid edges represent a parallel composition (Figure 2(b)). A model box with thickened solid edges indicates a separate executable image; in diagrams for which no model box of this type is present, the system is a single executable. Nesting of solid and dashed model boxes represents hybrid compositions; dashed (solid) model boxes represent serial (parallel) composition nested within a parallel (serial) composition (Figures 2(b) and (c)).

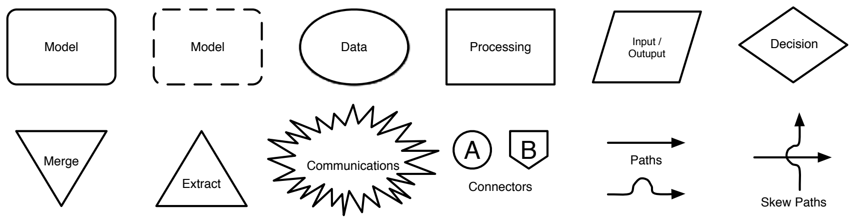


Fig. 1. Shapes of basic symbols used to diagram coupling workflows

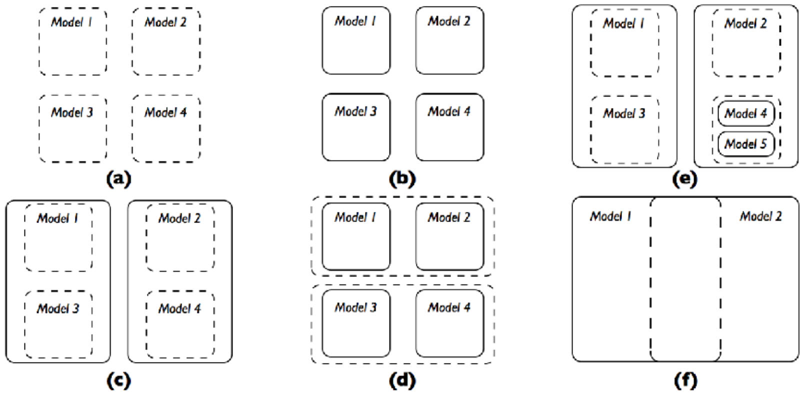


Fig. 2. Graphical conventions for representing process compositions: (a) serial composition, (b) parallel composition, (c) hybrid composition with parallel compositions embedded within a serial composition, (d) hybrid composition with serial compositions embedded within a parallel composition, (e) hybrid composition with multiple embedding levels, (f) overlapping composition

Multiple levels of nesting represent deeper levels of hybrid composition (e.g., Figure 2(e)). Intersecting boxes indicate overlapping composition (Figure 2(f)).

Coupling relies on the description of field and mesh data, and in the PCP on domain decomposition of these data. Figure 3(a) displays the pictograms for state, input, and output vectors, discretized domains and overlap regions, and domain decomposition objects. In some cases, it is convenient to incorporate field data with its resident mesh and domain decomposition in a simple object called a *bundle*[8], and a separate symbol is provided for this purpose. Pictograms for time- and space-integrated data objects are included. An "accumulator" is included for use either in place of the integrated field objects or to represent results of global reductions (e.g., MPI_REDUCE()) across a process pool. Not all of the data pictograms used in this paper are shown in Figure 3(a). Pictograms for state, coupling. and mesh data partitioned across a process pool are labeled by using notation from Section 2: $\{\mathbf{u}, \mathbf{v}, \mathbf{w}, \boldsymbol{\Delta}(\gamma), \boldsymbol{\Delta}(\omega)\}$ in place of $\{\mathbf{U}, \mathbf{V}, \mathbf{W}, \boldsymbol{\Delta}(\Gamma), \boldsymbol{\Delta}(\Omega)\}$. Also, for ease of hand-drawing of schematics, the bold-face notation on pictograms may be replaced with vector symbols; for example, $\vec{U}$ or $\underset{\sim}{U}$ in place of $\mathbf{U}$. Integrated data may also be represented by using triangular brackets—$\langle\mathbf{W}\rangle$, $\langle\mathbf{w}\rangle$, and so on.

Figure 3(b) displays the core set of pictograms used to depict intercomponent data movers $\mathscr{H}_{ij}$ and communications operations commonly used in couplers, including blocking- and nonblocking point-to-point messaging and $M \times N$ transfers, parallel data transposes, broadcasts, scatters, gathers, and synchronization. Other communications pictograms are created by annotating an empty burst symbol—for example, a global reduction.

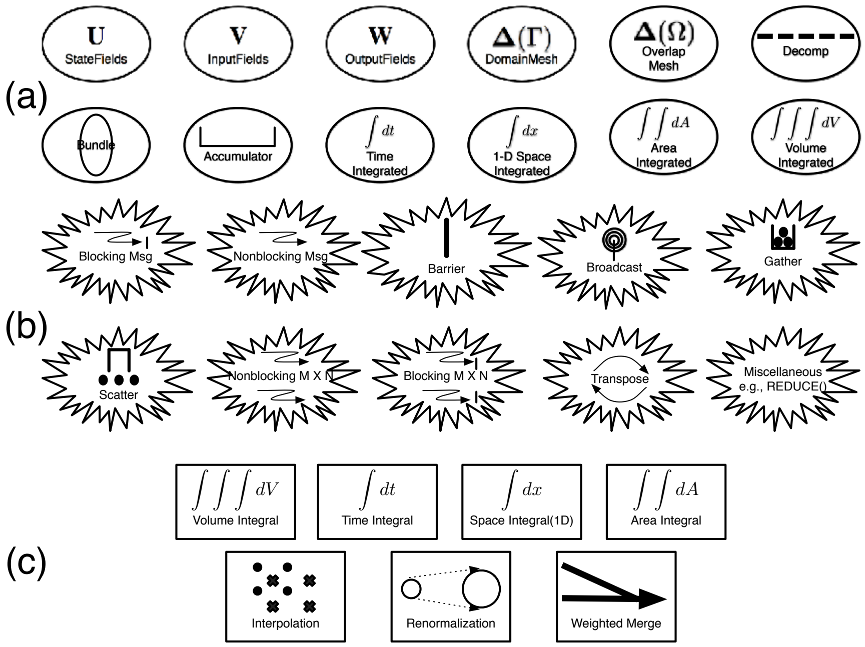


Fig. 3. Detailed pictograms for representing (a) data, (b) communications, and (c) data transformation

Figure 3(c) displays the core set of data transformation pictograms, including intergrid interpolation, temporal and spatial integration, renormalization to enforce conservation of flux integrals, and weighted merging.

4 Examples

A common coupled systems problem is intermodel data transfer. Figures 4 and 5 show the model-coupler communications patterns for versions 2 and 3 of the Community Climate System Model, respectively; see [8] for further details. Figure 4 depicts the communications between a model employing hybrid MPI/OpenMP parallelism and a coupler that is solely OpenMP parallel but possesses one MPI process. The models are in parallel composition, each implemented in a separate executable. Distributed output data **w** is gathered from the atmosphere's cohort to its root, yielding **W**. Nonroot processes wait in a barrier until the root finishes communicating with the coupler. A single blocking MPI message containing **W** is sent to the coupler, which receives it as input **V**. The coupler in turn posts a single blocking message to the atmosphere and continues processing upon completion. The atmosphere receives this message as **V**, and scatters the message to its processes, yielding **v**. Figure 5 depicts parallelized $M \times N$ data transfer between two separate executables, both possessing multiple MPI processes. The atmosphere sends its distributed output **w** in parallel to the coupler. Once the parallel send has completed, the atmosphere receives its distributed input **v** from the coupler. All communications are blocking overall; that is, an MPI_Waitall() is invoked to ensure that the model and coupler evolve together.

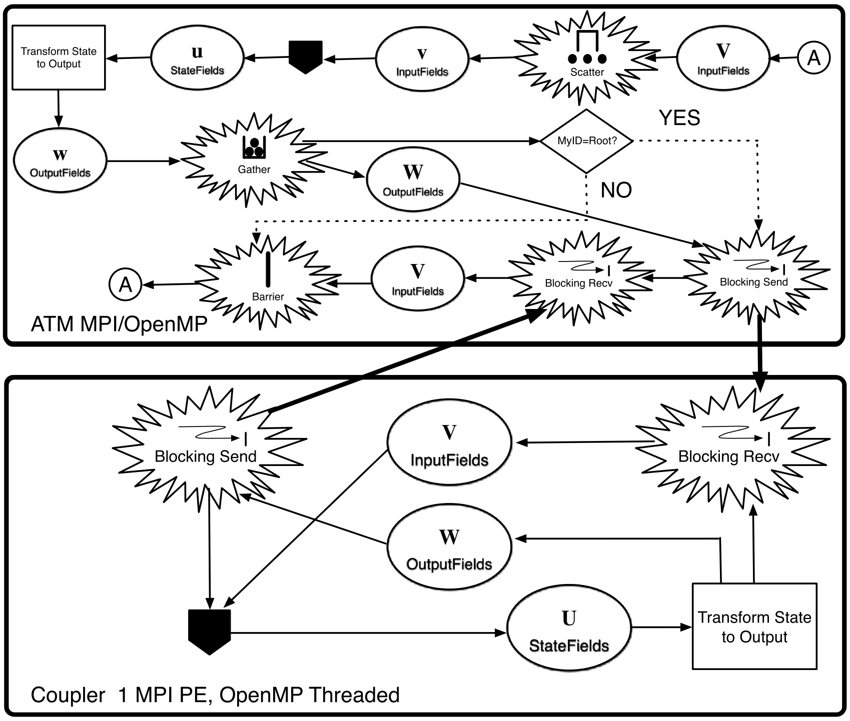


Fig. 4. Coupling workflow between two models featuring serialized communications

Three key types of data transformations used in coupling are shown in Figure 6. Figure 6 (a) depicts integrated data delivery processing. A model takes distributed instantaneous output **w** and integrates it with respect to time, accumulating fluxes and averaging state data. This process is performed over a coupling cycle period, and the decision box determines whether the model should continue or pause to couple to the rest of the system. Figure 6 (b) shows a scheme for enforcing conservation of interfacial fluxes under interpolation. Distributed input data **v** defined on Grid 1 are received and integrated across the two- dimensional boundary. The data are then interpolated to yield input defined on Grid 2; they are integrated on this grid over the boundary. The values of **v**, defined on Grids 1 and 2, together with their respective integrals, are then passed through a renormalization function, which computes the ratio of the integrals obtained on Grids 1 and 2 and uses their ratio to rescale the values of **v** residing on Grid 2, thus conserving global flux integrals across the boundary.

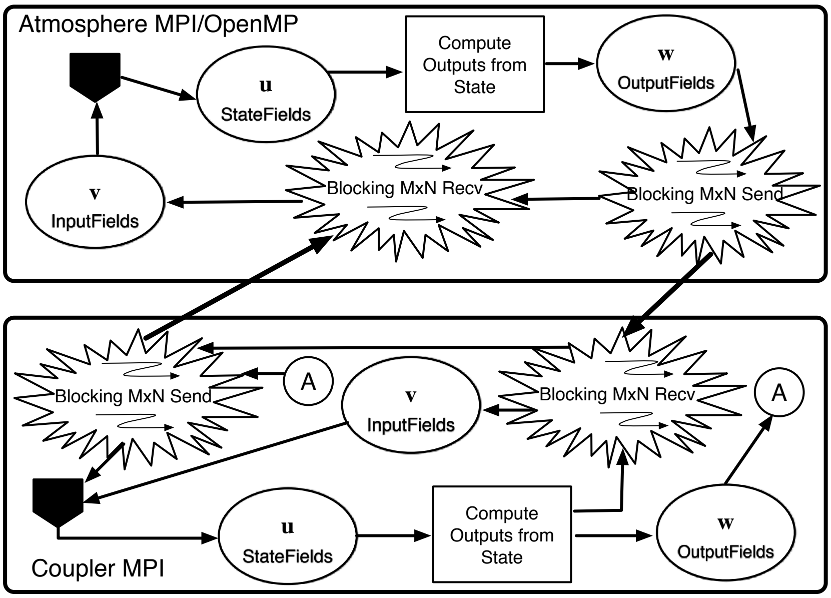


Fig. 5. Coupling workflow between two models featuring $M \times N$ transfer

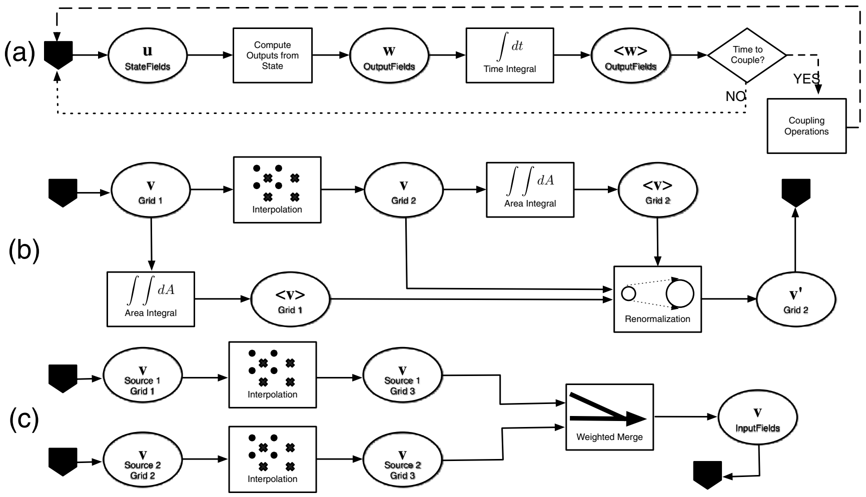


Fig. 6. Coupling workflows for (a) time-integrated data delivery, (b) flux-conserving interpolation, and (c) merging of inputs

5 Conclusions

I have proposed a graphical schema for describing intermodel data coupling that combines control and data flow. The schema is derived from a theoretical framework for describing intermodel coupling. The schema is rich enough to describe a wide variety of intermodel coupling situations, and user-extensible. I have employed the schema to depict commonly-encountered coupling workflows.

Associating the schema with software entities—for example, a generic coupling infrastructure toolkit such as the Model Coupling Toolkit (MCT) [9]—will enable graphical program specification for coupler mechanisms. This will require mappings between the schema and MCT's classes and methods, combined with code generation infrastructure, and is an exciting area for future investigation.

Acknowledgments. This work was supported by the US Department of Energy's Scientific Discovery through Advanced Computing program. Argonne National Laboratory is operated for the DOE by UChicago Argonne LLC under contract DE-AC02-06CH11357. I thank the Department of Theoretical Physics/ RSPhysSE at ANU for hosting me as a visiting fellow.

References

1. Larson, J.W.: Ten organising principles for coupling in multiphysics and multiscale models. ANZIAM Journal (accepted) (2008)
2. Dreyfuss, H.S.: Symbol Sourcebook: An Authoritative Guide to International Graphic Symbols. McGraw-Hill, New York (1972)
3. ANSI: Standard flowchart symbols and their use in information processing (X3.5). Technical report, American National Standards Institute, New York (1970)
4. ISO: Information processing–documentation symbols and conventions for data, program, and system flowcharts, program network charts and system resources charts. Technical report, International Organization for Standardization, Geneva (1985)
5. Yourdon, E.: Structured analysis wiki (2009), http://www.yourdon.com/strucanalysis/
6. OMG: Unified Modeling Language web site (2009), http://www.uml.org/
7. Bell, E.: Death by UML fever. ACM Queue 2(1), 72–80 (2004)
8. Craig, A.P., Kaufmann, B., Jacob, R., Bettge, T., Larson, J., Ong, E., Ding, C., He, H.: cpl6: The new extensible high-performance parallel coupler for the community climate system model. Int. J. High Perf. Comp. App. 19(3), 309–327 (2005)
9. Larson, J., Jacob, R., Ong, E.: The model coupling toolkit: A new fortran90 toolkit for building multi-physics parallel coupled models. Int. J. High Perf. Comp. App. 19(3), 277–292 (2005)

Photoabsorption and Carrier Transport Modeling in Thin Multilayer Photovoltaic Cell

František Čajko and Alexander I. Fedoseyev*

CFD Research Corporation, 215 Wynn Dr., Huntsville, AL 35805
ph.: (256)726-4928; fax: (256)726-4806
aif@cfdrc.com

Abstract. The paper describes an efficient implementation of a photoabsorption model for modern thin photovoltaic (PV) cells. As the modelling of solar cells constitutes a multiphysic and multiscale problem, a special attention was paid to the speed while retaining a reasonable accuracy. The model is restrained to a normal incidence but accounts for the whole solar spectrum. Applied transfer matrix method yields an accurate distribution of the light intensity in the semiconductor structure. Usage of equivalent parameters makes it possible to simulate both plain semiconductor material and a quantum dot superlattice based material, which is used to enhance the PV device performance.

Keywords: photovoltaic cell, photoabsorption, carrier generation, multiple wavelength.

1 Introduction

Solar cells answer a call for alternative energy by providing a source of a reliable, long lasting power supply. Most designs of solar cells rely on semiconductor materials based on silicon or III-V semiconductors. Space electronic equipment requires improvements in solar cell efficiency to deliver more efficient, lightweight solar cells. Recently proposed approaches to enhance the efficiency utilize the novel nanomaterials containing quantum dots [1], and new concepts of cell design. A proper models and simulation techniques are needed to speed-up the development on novel solar cell devices and reduce the related expenses.

Modeling of a photoabsorption in solar cells is a complex problem that involves multiphysics simulations of multiscale problem at different levels [2]. Recent trend of using quantum dots to enhance the performance of a solar cell by broadening an absorption spectrum and reducing a recombination rate requires modeling at a quantum level. A spatial distribution of the light intensity in the structure is obtained at a macroscopic level because of the dimensions of the solar cell. Carrier generation and recombination rates are used in transport equations in order to calculate a carrier distribution.

Our goal is to develop a simulation tool NanoTCAD [3] that will accurately predict a performance of a solar cell and allows an inside view that helps in future

* Corresponding author.

designs. The NanoTCAD simulator already implements transport equations in semiconductor devices. It can compute a steady-state or a transient. Because of solar cells we are now interested more in the steady-state. A simulation of a typical design takes 100 – 600 thousands volume elements. For a simulation of a small problem without details with only 8,000 elements the computation on 2GHz Pentium PC takes about 25 seconds and such simulation represents only one point on a I-V curve. Therefore CPU time saving is highly desirable, especially in time dependent problems, e.g. a radiation damage modelling.

This paper is focused on a simplified and yet accurate model of photoabsorption to be included into the simulator. The essential part is to compute an electromagnetic field (EM) distribution inside the solar cell. We want to achieve this without a necessity to implement a full electromagnetic solver that would substantially slow down the 3D computation.

2 Geometry of Solar Cells

We consider a sandwitch structure of a solar cell with upper electrodes organized in a grid and creating rectangular windows for the light to enter into the structure. The incident solar light is a plane wave limited by the window between electrodes impinging the top surface along the normal. A part of the solar cell is depicted in Fig. 1. The aperture defines a horizontal cross section of the beam. It is assumed that the cross section of the beam doesn't vary in the vertical direction due to a shallow geometry of solar cells. Formally, the horizontal area can be split into three parts: (i) the area under the window where a plane wave penetrating in a vertical direction is assumed – carrier transport and photoabsorption occur, (ii) the area outside the window with no light intensity – electron transport but no photoabsorption, (iii) area near the window edge – diffraction and scattering take place. This model is suitable for solar cells because the window is much larger than the wavelength so that diffraction and scattering effects on a border of the cross section can be neglected.

The light propagating through the window of a solar cell can be decomposed into periodic fields

$$E(x,y,z) = \int \tilde{E}(k_x,k_y;z)e^{i\frac{2\pi}{\lambda_0}(k_x x + k_y y + \beta z)}dk_x dk_y \tag{1}$$

with value

$$\beta = \sqrt{\varepsilon_r - k_x^2 - k_y^2} \tag{2}$$

It is obvious that for $(k_x^2 + k_y^2) << \varepsilon_r$, the value of β is almost constant, meaning that all waves propagates at about the same phase velocity, and hence, the horizontal profile is changing very slowly. The relative difference of the electric field caused by neglecting k_x, k_y in the expression above is

$$\left|\frac{E - E(k_x = k_y = 0)}{E(k_x = k_y = 0)}\right| = |e^{i\frac{2\pi}{\lambda_0}(\beta - \sqrt{\varepsilon_r})z} - 1| \approx \left|\frac{\pi(k_x^2 + k_y^2)z}{\lambda_0\sqrt{\varepsilon_r}}\right| \tag{3}$$

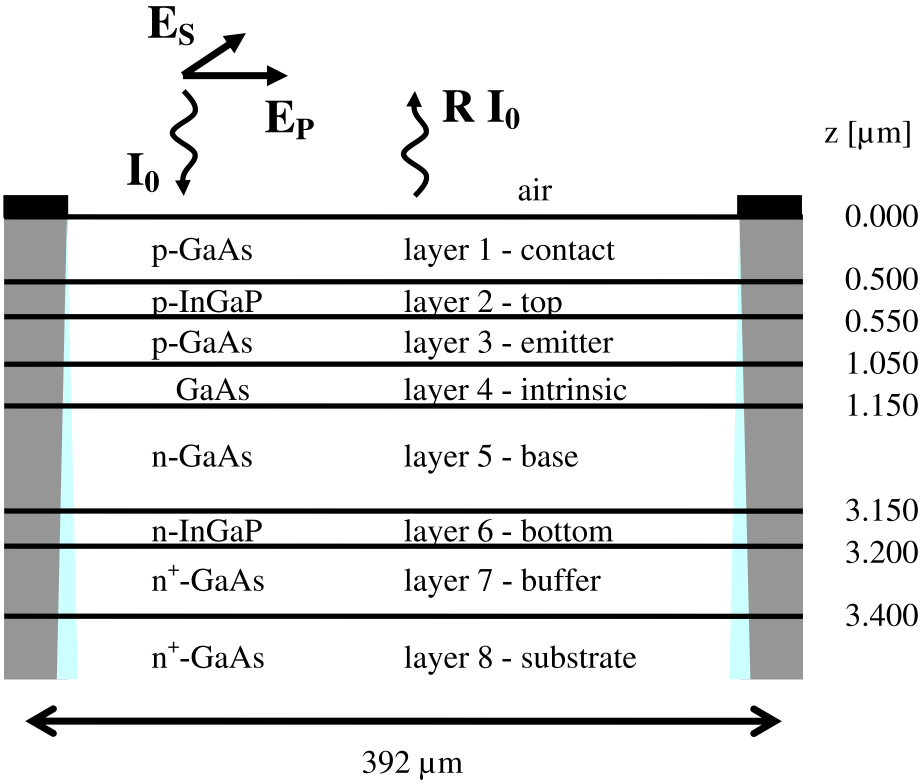


Fig. 1. A schematic multilayer structure with a window for incident light created by electrodes (black). Incident irradiation I_0, reflected power flux RI_0 and two polarizations E_P, E_S of the electric field shown. Shaded areas respresents regions without light.

for components with small k_x and k_y. Considering a typical value of $\sqrt{\varepsilon_r}$ to be 3, it is required that $(k_x^2 + k_y^2)z << \lambda_0$. For an aperture with a diameter a the components that matter have wavenumbers up to $2\pi/a$ (i.e. $\sqrt{(k_x^2 + k_y^2)} = \lambda_0/a$). For a structure with a height about λ_0 and a window diameter about 1 cm the estimated relative error in the center of a window that corresponds to one Fourier component is about 10^{-4}.

3 Plane Waves in the Structure at a Single Wavelength

All properties within one layer are assumed to be constant but dependent on a free space wavelength λ_0. Electrooptical properties of each layer are described by a refractive index n and an extinction coefficient k. A general solution in the j^{th} homogeneous layer is a superposition of two counterpropagating waves

$$\boldsymbol{E}_j(z) = \boldsymbol{E}_{0j}^{+} e^{+iK_j(z-z_{0j})} + \boldsymbol{E}_{0j}^{-} e^{-iK_j(z-z_{0j})} \tag{4}$$

with $e^{-i\omega t}$ time factor. The complex vectors $\boldsymbol{E}_{0j}^{+}$ and $\boldsymbol{E}_{0j}^{-}$ represent the values of plane wave components in a reference depth $z = z_{0j}$ (chosen at the top of the layer j) propagating downwards and upwards, respectively. Each propagating wave is a solution of the Helmholtz equation

$$\nabla^2 \boldsymbol{E}_j(z) + K_j^2 \boldsymbol{E}_j(z) = 0 \tag{5}$$

with a wavenumber

$$K_j = K_0(n_j + ik_j) = \frac{2\pi}{\lambda_0}(n_j + ik_j) \tag{6}$$

Given the assumption about geometry and the normal direction of incidence, the EM fields vary only along the normal. Any solution of such problem can be decomposed into two independent polarizations (Fig.1) — S ($E = \hat{y}E_y, H = \hat{x}H_x$) and P ($E = \hat{x}E_x, H = \hat{y}H_y$) — that are just a mutual rotation and both are described by the same Helmholtz equation (5).

Natural light is not polarized. It means an equal contribution of both polarizations. As we are interested in the power intensity (proportional to $|\boldsymbol{E}|^2 = |E_x|^2 + |E_y|^2$) and solutions for both polarizations are identical, the formula

$$E(z) = E_0^{+} e^{+iK(z-z_0)} + E_0^{-} e^{-iK(z-z_0)} \tag{7}$$

for a scalar equivalent field intensity $E = |\boldsymbol{E}|$ can be used instead. The overall solution is tailored by applying a continuity boundary conditions across the layer interfaces for tangential components of the electric and magnetic fields assuming one of the polarizations.

The usual approach is to use the transfer matrix method [5]. An electric field at each point is represented by two complex fields E^{+} and E^{-}. In a homogeneous layer they are given by the subexpressions of (7) or in a matrix form

$$\begin{pmatrix} E^{+}(z) \\ E^{-}(z) \end{pmatrix} = \begin{pmatrix} e^{+iK(z-z_0)} & 0 \\ 0 & e^{-iK(z-z_0)} \end{pmatrix} \begin{pmatrix} E^{+}(z_0) \\ E^{-}(z_0) \end{pmatrix} \tag{8}$$

A transition across an interface between two layers is defined by boundary conditions for the electric and magnetic fields

$$\begin{pmatrix} 1 & 1 \\ 1/Z_1 & -1/Z_1 \end{pmatrix} \begin{pmatrix} E_1^{+} \\ E_1^{-} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1/Z_2 & -1/Z_2 \end{pmatrix} \begin{pmatrix} E_2^{+} \\ E_2^{-} \end{pmatrix} \tag{9}$$

where Z_i is an intrinsic impedance of the layer i with refraction index n_i and extinction coefficient k_i and relates to the intrinsic impedance Z_0 of vacuum as

$$Z_i = \frac{Z_0}{n_i + ik_i} \approx \frac{377\Omega}{n_i + ik_i} \tag{10}$$

One can find that

$$\begin{pmatrix} E_1^{+} \\ E_1^{-} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 + Z_1/Z_2 & 1 - Z_1/Z_2 \\ 1 - Z_1/Z_2 & 1 + Z_1/Z_2 \end{pmatrix} \begin{pmatrix} E_2^{+} \\ E_2^{-} \end{pmatrix} \tag{11}$$

The total electrical field can be reconstructed by assuming no backward propagating wave in the bottom layer (i.e. $E_0^- = 0$) and a unit amplitude of the forward wave ($E_0^+ = 1$). Then, applying matrix operations (8),(11) yield a recursive formulas for layers above

$$E_{0,j}^+ = (aE_{0,j+1}^+ + bE_{0,j+1}^-) \cdot exp[+iK_j(z_{0,j} - z_{0,j+1})] \tag{12}$$

$$E_{0,j}^- = (bE_{0,j+1}^+ + aE_{0,j+1}^-) \cdot exp[-iK_j(z_{0,j} - z_{0,j+1})] \tag{13}$$

with coefficients $a = 1 + Z_j/Z_{j+1}$ and $b = 1 - Z_j/Z_{j+1}$. Once all amplitudes are known, they are scaled in order to match the actual incident irradiance I_0, i.e. the forward propagating wave in air must be

$$E_{\text{air}}^+ = \sqrt{2Z_0 I_0} \tag{14}$$

The transfer matrix method calculates the reflectance of the structure for an ideal case. Since the real parameters of semiconductors are not exactly known, and because of a surface roughness and scattering from edges of the window, the experimentally measured reflectance can differ from the calculated one. Fortunately, there is an easy calibration.

Let I_u be the incident irradiance and R_u a measured reflectance supplied by a user. Then, power $(1 - R_u)I_u$ transmitted into the structure should be preserved. If the calculated reflectance from the first layer is R_0, then the equivalent irradiance used in the calculations is

$$I_0 = I_u \frac{1 - R_u}{1 - R_0}. \tag{15}$$

4 Carrier Generation Rate

Linear photoabsorption refers to a phenomenon of capturing a photon that causes a generation of one electron-hole pair (e-h). It can be described by the classical electrodynamics or by the quantum mechanics. In the classical electrodynamics the interaction with material is described by a complex permittivity, or alternatively, by a refraction index and an extinction coefficient. At this level we are able to calculate a light intensity distribution in the material which is important for an energy absorption. The absorbed energy is used for an electron transition and, hence, links both, classical and quantum mechanical, approaches together.

The total carrier generation rate is contributed by all spectral components for which a photoabsorption occurs. First, an expression for a single frequency will be found, then an integration over the spectrum will be discussed. All calculations assume one dimensional layer structure where the field distribution is well known.

4.1 Single Wavelength

The horizontal profile of the light intensity is well defined. The absorbed power per unit volume can be calculated directly as

$$\frac{dP_{\text{ab}}}{dV} = \frac{\omega\varepsilon_0\varepsilon''}{2}|E|^2 = 2nkK_0\frac{|E|^2}{2Z_0} \tag{16}$$

where $\omega = K_0 c = 2\pi c/\lambda_0$ is an angular frequency of the light, c speed of the light in vacuum, $n + ik$ the complex refraction index, Z_0 the intrinsic impedance of vacuum and

$$|E|^2 = |E^+(z) + E^-(z)|^2 \tag{17}$$

The electric field $(E^+ + E^-)$ is given by (7). The amplitudes E_0^+, E_0^- and the reference position z_0 relate to a layer at the given position z. It is easy to show that

$$|E|^2 = |E_0^+|^2 e^{-2kz'} + |E_0^-|^2 e^{+2kz'} + 2\Re\left\{E_0^+(E_0^-)^* e^{i2nz'}\right\} \tag{18}$$

To save space, a shorted the notation

$$z' = K_0(z - z_0) \tag{19}$$

was used. The value of K_0 is defined by (6) and the asterisk $(^*)$ denotes a complex conjungated value.

Our simulation tool NanoTCAD uses a volume element method. The following expression is the power P_{ab}^{cell} absorbed in one cell of the volume element method for given wavelength λ_0 of the light

$$P_{\mathrm{ab}}^{\mathrm{cell}} = \int_{\mathrm{cell}} \frac{dP_{\mathrm{ab}}}{dV} dV = A \int_{z_1}^{z_2} \frac{dP_{\mathrm{ab}}}{dV} dz \tag{20}$$

The integration over x and y can be taken care of by a simple multiplication by a horizontal cross sectional area A of the cell. The integration along z has to be conducted properly. Knowing positions of the top and bottom walls of the cell, denoted by z_1 and z_2 $(z_1 < z_2)$ and the primitive function of (17) which is

$$\frac{1}{2K_0 k}\left[-|E_0^+|^2 e^{-2kz'} + |E_0^-|^2 e^{+2kz'}\right] + \Im\left\{\frac{E_0^+(E_0^-)^*}{K_0 n} e^{i2nz'}\right\} \tag{21}$$

the absorbed power can be expressed as

$$P_{\mathrm{ab}}^{\mathrm{cell}} = \frac{An}{2Z_0}\left[|E_0^-|^2 e^{+2kz'} - |E_0^+|^2 e^{-2kz'} + 2\frac{k}{n}\Im\left\{E_0^+(E_0^-)^* e^{i2nz'}\right\}\right]_{z'=K_0(z_1-z_0)}^{K_0(z_2-z_0)} \tag{22}$$

The refraction index n for ordinary materials is non-zero, so the expression is well defined. Once the power absorbed in a cell is calculated, the electron-hole generation rate in the cell can be obtained as [4]

$$dN_{\mathrm{e-h}}/dt = \frac{\eta}{\hbar\omega} P_{\mathrm{ab}}^{\mathrm{cell}} \tag{23}$$

where the internal quantum efficiency η is a property of the material and $\hbar\omega$ is energy of photons at a given wavelength. The equation represents the total generation rate of the whole volume element at the given wavelength.

4.2 Spectrum of the Solar Light

The solar radiation outside the Earth's atmosphere is described by air mass 0 (AM0) spectrum. The total amount of delivered power is known as a solar constant S. Its value was determined to be $1366.1 W/m^2$ [6]. We use 2000 ASTM Standard Extraterrestrial Spectrum Reference (E-490-00) obtained from [6] for simulation of solar cells. Abrupt (noise like) changes in the spectrum (Fig.2) can prevent an observation of a convergence of a numerically evaluated spectral integral because of a random choice of sample points. Therefore a cumulative spectral density Φ was introduced. An irradiance in interval of energies $E_1 \ldots E_2$ is then

$$d\Phi(E_1, E_2) = |\Phi(E_1) - \Phi(E_2)| \tag{24}$$

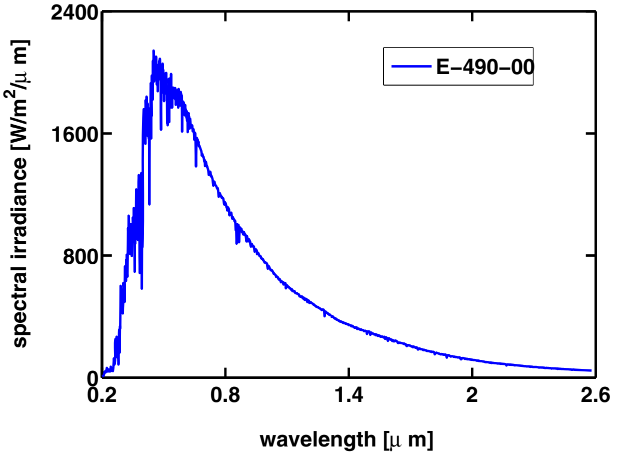


Fig. 2. Part of the AM0 spectrum. Data from E_490_00.

4.3 Multiple Wavelengths

Since the system is linear, the field distribution (7) can be calculated under an assumption of a unit incident irradiance I_0 yielding the absorbed power within one cell at the specified frequency calculated from (22) to be P_1^{cell}. Then, it is multiplied by a true incident power within a small interval of energies dE

$$dP_{\mathrm{ab}}^{\mathrm{cell}} = P_1^{\mathrm{cell}} \frac{d\Phi(E)}{dE} dE = P_1^{\mathrm{cell}} d\Phi(E) \tag{25}$$

For a multicolor light, the e-h pair generation rate is an integral of the expression (23) over the whole spectrum of the incident light (e.g. AM0)

$$dN_{\mathrm{e-h}}^{cell}/dt = \int \eta(E) \frac{P_1^{\mathrm{cell}}(E)}{E} \frac{d\Phi(E)}{dE} dE \tag{26}$$

Notice that also P_1^{cell} has to be recalculated for every λ according to (22). There are two reasons for that: (i) material properties are frequency dependent and (ii) optical distance (phase difference) between layers changes as well.

The integral (26) will be evaluated numerically. The lower limit is given by the lowest energy with a nonzero quantum efficiency η. The upper limit is given by the spectrum itself. We choose to cut-off the spectrum at 99% of the total energy.

The integration of (26) can be performed in a semianalytical way: (i) the spectrum is split into M subintervals, (ii) an analytical integration is performed in each subinterval, (iii) partial results are summed together

$$dN_{\mathrm{e-h}}^{\mathrm{cell}}/dt = \int_{\hbar\omega_{\mathrm{START}}}^{\hbar\omega_{\mathrm{STOP}}} G_1^{\mathrm{cell}} d\Phi = \sum_{i=0}^{M-1} \int_{\hbar\omega_i}^{\hbar\omega_{i+1}} G_1^{\mathrm{cell}} \frac{d\Phi}{dE} dE \tag{27}$$

Here, $G_1^{cell} = \eta(E)P_1^{cell}(E)/E$ is a generation rate in a cell for a monochromatic light with a unit incident irradiance, $\hbar\omega_i$ are the split points and E is an integration variable that runs over all photon energies $\hbar\omega$.

In order to do the analytical integration on subintervals the following approximations were made

1. the spectral density $d\Phi/dE$ is constant on each subinterval
2. the generation rate can be approximated by a piecewise linear function

$$G_1^{\mathrm{cell}}(\hbar\omega) \approx G_1^{\mathrm{cell}}(\hbar\omega_i) + \frac{G_1^{\mathrm{cell}}(\hbar\omega_{i+1}) - G_1^{\mathrm{cell}}(\hbar\omega_i)}{\hbar\omega_{i+1} - \hbar\omega_i}(\hbar\omega - \hbar\omega_i) \tag{28}$$

This approach results in a simple expression

$$dN_{e-h}^{\mathrm{cell}}/dt = \sum_{i=0}^{M-1} \frac{G_1^{\mathrm{cell}}(\hbar\omega_i) + G_1^{\mathrm{cell}}(\hbar\omega_{i+1})}{2} d\Phi(\hbar\omega_i, \hbar\omega_{i+1}) \tag{29}$$

5 Numerical Results and Discussion

The modeling of 3D electromagnetic field calls for accurate models of semiconductor properties. The refraction and extinction coefficients necessary to calculate an absorption rate are usually obtained by the ellipsometry technique and can be found in the literature [7,8].

The model was implemented into Matlab$^{\mathrm{TM}}$ for testing purposes. 99% of the AM0 spectrum was considered in the computation. The spectrum was further cut-off at the longest wavelength at which a photoabsorption occurs yielding a range of photon energies 1.375 eV – 4.16 eV. This range was divided into M subinterval. An convergence of the total deposited charge was observed and $M = 100$ chosen based on 1% error. A distribution of a e-h generation rate G was computed on a standard 3 GHz PC within a fraction of second. Unlike in the NanoTCAD implementation, the generation rate was calculated along a

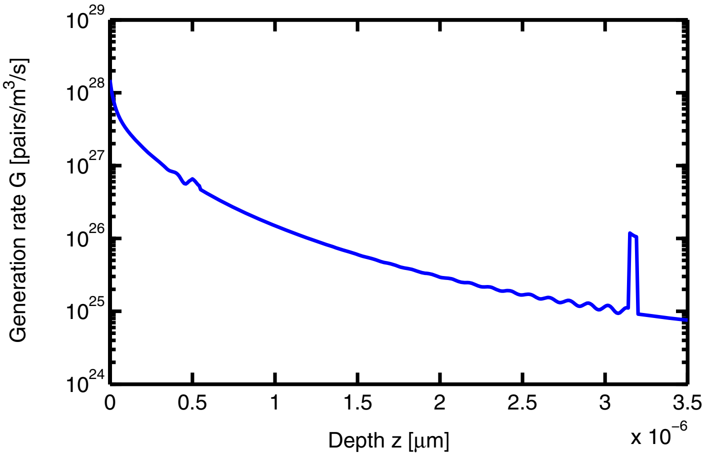


Fig. 3. A carrier generation rate in the solar cell structure along a vertical axis in the center of the window

vertical axis only in the center of the window. The actual implementation has to compute G for every cell of the volume element method.

It is not possible to calculate the generated photocurrent without solving transport equations. Therefore only its order was calculated assuming 60% external efficiency [4] and compared with a typical value of $10\,\mathrm{mA/cm^2}$ for GaAs solar cell.

The testing model comprises of a structure in Fig. 1. The cumulative generation rate per unit cross section saturates at 1.46×10^{21} pairs/m^2/s what represent 100% external quantum efficiency. Since most of the structure is created from GaAs a factor of 0.6 was used to account for the external quantum efficiency, thus yielding 0.85×10^{21} pairs/m^2/s or $13.6\,\mathrm{mA/cm^2}$ which is in the expected order.

For illustration purposes a distribution of a volume generation rate G is displayed in Fig. 3. In each material the generation rate decays exponentially with a corresponding attenuation constant. The wiggles observed between 2 and $3.15\,\mu m$ are due to an interference of forward and backwards waves. It seems that the main source of the reflection is the bottom InGaP layer that manifest itself by a 50 nm wide spike of increased light absorption. Similar behavior of a smaller relative amplitude is observed above the top InGaP layer.

6 Conclusions

A photoabsorption model for solar cells and technical details of the computation were presented. Extra assumptions were made for solar cell applications: the structure irradiated by a plane wave, a size of the solar cell window much larger than the wavelength, scattering on the edges of the window and on inhomogeneities as well as a scattering due to a photon recycling neglected. The material properties were described by an index of refraction and an extinction coefficient. In this paper are considered to be known.

The described approach resulted in a Matlab^{TM} implementation of the photoabsorption model for testing purposes and yielded preliminary results. An absorption rate for one test case is presented in the previous section. The e-h generation is mostly defined by the material properties. The total deposited charge of 13.6 mA/cm^2 is found to be in the expected order of magnitude.

The full implementation into NanoTCAD is still an ongoing work. The test cases include plain binary and ternary III-V semiconductors but the model is intended for quantum dots superlatices (QDS) as well. Equivalent parameters of QDS have to be calculated separately.

Acknowledgments. This work has been supported by AFOSR under Contract No. FA9550-07-C-0059 and by NASA under Contract No. NNC07CA20C.

References

1. Shao, Q., Balandin, A.A., Fedoseyev, A.I., Turowski, M.: Intermediate-band solar cells based on quantum dot supracrystals. App. Phys. Let. 91(16) (2007)
2. Fedoseyev, A.I., Turowski, M., Wartak, M.S.: Kinetic and Quantum Models in Simulation of Modern Nanoscale Devices. In: Balandin, A.A., Wang, K.L. (eds.) Handbook of Semiconductor Nanostructures and Devices. American Scientific Publishers, Los Angeles (2006)
3. CFDRC, NanoTCAD web site (2008), `http://www.cfdrc.com/bizareas/microelec/micro_nano/`
4. Sze, S.M.: Physics of Semiconductor Devices, 2nd edn. John Wiley & son, New York (1981)
5. Piprek, J.: Semiconductor Optoelectronic Devices: Introduction to Physics and Simulation. Academic Press, New York (2003)
6. 2000 ASTM Standard Extraterrestrial Spectrum Reference E-490-00, Renewable Resource Data Center, National Renewable Energy Laboratory, `http://rredc.nrel.gov/solar/spectra/am0/ASTM2000.html`
7. Palik, E.D. (ed.): Handbook of Optical Constants of Solids. Academic Press, New York (1985)
8. Adachi, S.: Optical dispersion relations for GaP, GaAs, GaSb, InP, InAs, InSb, $Al_xGa_{1-x}As$, and $In_{1-x}Ga_xAs_yP_{1-y}$. J. Appl. Phys. 66, 6030–6040 (1989)

Towards Multiscale Simulations of Carbon Nanotube Growth Process: A Density Functional Theory Study of Transition Metal Hydrides

Satyender Goel[1] and Artëm E. Masunov[1,2]

[1] Nanoscience Technology Center, Department of Chemistry
[2] Department of Physics,
University of Central Florida, 12424 Research parkway, Suite 400, Orlando, FL 32826 USA
amasunov@mail.ucf.edu

Abstract. Nanoelectronics and photonics applications of single wall carbon nanotubes (SWNT) are feasible only if SWNTs have specific chirality. The knowledge of the detailed mechanism for SWNT synthesis would allow one to optimize the chemical vapor deposition (CVD) process and may help to gain control over selectivity of SWNT synthesis. While it is not probably feasible to study this mechanism experimentally, it could be analyzed using molecular simulations. Here we propose multiscale computer modeling of CVD process. High theory level can be used for di- and tri-atomic fragments, in order to generate parameters for bond order force field. In turn, force field simulations will be used to characterize the chemical origin and thermochemical properties of the intermediates and transition states. This will allow predicting the rate constants for the elementary steps, which are then used in kinetic Monte Carlo simulations to describe SWNT growth at realistic time scales.

1 Introduction

Single wall carbon nanotubes (SWNT) are cylindrical molecules with unique properties. Many potential applications have been proposed for carbon nanotubes, including conductive and high-strength composites; energy storage and energy conversion devices; sensors; field emission displays and radiation sources; hydrogen storage media; and nanometer-sized semiconductor devices, probes, and interconnects. Some of these applications are now realized in products, others are demonstrated in prototypes. One of the difficulties for nanoelectronic applications of single-walled nanotubes is efficient separation of the nanotubes of different chirality. Increasing chiral selectivity of SWNT synthetic process could solve this problem.

There are successful attempts for selective synthesis of SWNT [1]. Lolli et. al. found that using a CoMo catalyst the chirality distribution of the groeing SWNTs can be reproducibly altered by varying the reaction temperature, the gaseous feed, or the cluster surface morphology. Specifically, increasing the temperature results in increase in nanotube diameter, without a change in the chiral angle. In contrast, by changing the support from SiO_2 to MgO, SWNT with similar diameter but different chiral angles are obtained. Clearly, different chirality distributions obtained when varying catalysts support or reaction conditions demonstrate that it is the result of chirality-specific differences in the growth kinetics, which in turn depends on the nanotube cap-metal cluster interaction. To fully control SWNT synthesis, one needs guidelines for optimizing the catalyst structure and experimental conditions, which are difficult to obtain without the detailed knowledge of the mechanism for SWNT catalytic synthesis.

Computer simulations may be helpful to establish this mechanism. However, only a few examples are found in the literature that attempt to simulate the SWNT growth. Reactive empirical bond order (REBO) force field was employed to model the catalyzed growth of nanotubes by CVD and investigates nanotube stability as a function of nanotube type, length and diameter through molecular dynamics (MD) approach [2], Since REBO parameters are not available for metals, no catalyst nanoparticles (NP) were considered in this study. The effect of iron cluster size on the structure defects and diameter of the SWNT was reported [3]. Specifically, the study showed that for large particles, containing at least 20 Fe atoms, the caps grow in diameter until they have the same diameter as the cluster. Unfortunately, artificially fast growth rate that made MD simulations feasible, resulted in formation of extremely defective nanotubes. Therefore, it is of interest to simulate catalytic SWNT growth using kinetic models, rather than direct MD approaches. The free energies of the intermediates and reaction barriers need to be obtained, then kinetic Monte Carlo (MC) modeling can be used to determine the effect of reaction conditions on SWNT morphology.

Here we describe multiscale approach to SWNT growth simulation. First, hybrid DFT is used for di- and tri-atomic fragments, to generate parameters for reactive bond order force field. At the next step SWNT/nanocluster systems are constructed, and their geometry is optimized for initial and final structures for different steps of SWNT growth. These structures are then used in MD simulations to identify additional reaction intermediates necessary to build a kinetic model for stepwise SWNT growth. We will start with SWNT having N hexagonal carbon rings in it. The SWNT growth process towards the addition of next ring has number of possibilities through different intermediates with varied reaction rates. Rate constant *k* for each reactive step will be calculated using Arrhenius equation (1).

$$k = Ae^{-E_a / RT} \tag{1}$$

Here *A* is the pre-exponential factor, *R* is the gas constant, *T* is the temperature, and E_a is the activation energy. that is necessery for the reactive system to cross the barrier from one intermediate to the other. A transition states is defined as the state corresponding to the highest energy along this reaction coordinate and is always a first-order saddle point in the energy map. Each transition state can be determined computationally with normal mode analysis by following a specific reaction coordinate corresponding to a single imaginary frequency. When the rate constants of the intermediate formation are known, the kinetic model for catalytic growth can be developed and implemented in Kinetic Monte-Carlo code. Finally, the resultant growth rates calculations using kinetic Monte-Carlo technique will be repeated for SWNT of different chirality. The developed protocol will be used to study SWNT growth with different catalysts under different temperature and feed rate conditions. The specific combinations of these conditions optimized for maximum selectivity will be selected for experimental verification.

Here, we have investigated bond dissociation energetics for different spin multiplicities in gas-phase neutral hydrides, formed by *3d*-transition metals from Sc to Cu. Broken-symmetry approach was adapted in order to get the qualitatively correct description of the bond dissociation. The resultant calculations from the current study will be used to parameterize REBO force field for transition metals.

2 Theoretical Methods

Density functional theory (DFT) [4, 5] in Kohn-Sham formalism combined with approximate exchange-correlation functionals [6] has become a method of choice for the calculation of numerous properties of molecules and solids. Advantage of DFT is considerably lower computational cost as compared to high level multireference *ab-initio* methods of Wave Function Theory (WFT) [7]. Unlike force-field approaches, DFT does not need tedious empirical parameter fitting to produce acceptable results.

In the past decade Transition Metal (TM) hydrides had been used as benchmark system to study efficiency of Density Functional Theory methods. Barone et. al. used pure and hybrid DFT functionals BLYP and B3LYP to study transition metal complexes which includes first row TM hydrides and their cations [8]. B3LYP was found to give accurate dissociation energies, but somewhat overestimate the bond lengths and dipole moments. In a detailed study of *3d* transition metal systems including monohydrides Furche and Perdew [9] were not able to reproduce these dissociation energies with the same functionals and basis sets. Presumably, their SCF procedure systematically converged to a different local minimum, as symmetry-adapted (SA) unrestricted Kohn-Sham formalism (UKS) was used by Borane et. al. and Broken Symmetry orbitals were used by Furche and Perdew. Among various semilocal (LSDA, BP86, PBE, TPSS) and hybrid density functionals (B3LYP, hTPSS), Furche and Perdew recommend functional TPSS as workhorse of TM compounds. Jensen et. al. [10] has investigated performance of five different density functionals (B3LYP, BP86, PBE0, PBE, BLYP) for diatomics of first row transition metal systems. They concluded that success of a functional is system specific, which means all of these functional are more accurate for certain system and less accurate for others.

Wavefunction theory (WFT) methods were also employed repeatedly to study TM hydrides. WFT uses different approximations to exact solution of the Schrodinger equation given by full configuration interaction (CI) method. Most of them involve multireference SCF procedure (CASSCF or MCSCF) to treat the static correlation, supplemented by double excitations (SDCI or SOCI) to account for dynamic electron correlations. The electronic structure of NiH was investigated using single reference CI methods three decades ago [11] and, more recently with multireference methods with or without relativistic effects [12-14]. Potential energy curves had been also calculated for other first row TM hydrides, including TiH [15-17], CoH [18], CuH [14, 19], VH [20], and ScH [21]. Bauschlicher et al. [20] studied the first row TM hydrides (TiH, VH, CrH, MnH, FeH, NiH) using CASSCF/SDCI method. Related method MCSCF+SOCI was used by Koseki et al. to study both ground and excited state PECs of the five first row TM hydrides (ScH, TiH, VH, CrH, MnH) recently [22-24].

Another approximation to full CI was taken by Reiher et. al. [25] to study CoH. They used density matrix renormalization group (DMRG) technique that allows performing CI iteratively without the need to explicitly store any long Slater determinant expansion. DMRG is making study of larger systems involving transition metals feasible, and accurately predict the energy gaps between different spin states.

In this contribution we study diatomic transition metal hydrides, using two DFT functionals and compare their accuracy to the experiment, and WFT results.

3 Computational Details

All calculations were done with Gaussian03 program [26] using all-electron Wachters+f basis set [27, 28]. Spin-polarized (unrestricted) DFT was used throughout with no spatial symmetry constraints (broken symmetry, BS). Initial guess was generated by using Harris functional [29] which is the default option in Gaussian03. Self-consistent field (SCF) convergence threshold was set to 10^{-7}, and relaxed to 10^{-5} in a few problematic cases. Initial guess was followed by either geometry optimization or scan along interatomic distance to plot the potential energy curve. In some cases (CrH, VH) Harris guess lead to SCF convergence problems, and Hartree-Fock (HF) orbitals were used as a guess. In a few cases where geometry optimization was terminated due to convergence failure, and converged KS orbitals were used as initial guess (BMK orbitals in case of TPSS non-convergence and vice versa).

A potential complication in the study of systems with nearly degenerate energy levels is the danger of obtaining distinctly different SCF solutions as local energy minima. When different solutions are obtained for the equilibrium geometry and for the dissociation limit, the energy difference is no longer physically meaningful. In order to ensure consistency of SCF solution for all geometries, we built entire potential energy curves and verified that it does not have discontinuities indicating the switch from one SCF solution to another. SCF process was started with fractional occupation numbers (FON) around the Fermi level by using keywords SCF=Fermi and IOp(5/22=5). The occupational numbers became integer at the final SCF cycles. The stability of the SCF solution was checked and KS orbitals were re-optimized (if unstable) using keyword Stable=Opt.

Molden [30] graphical interface was used to examine Kohn-Sham orbitals at the dissociation limit, and at the points where potential curves were found to be non-monotonic. For one case (CrH) in order to obtain SCF solution with the lower energy, spin-polarization of σ-bond had to be inverted to have minority spin density localized on H atom using the keyword Guess=Alter.

4 Results and Discussion

4.1 Dissociation Energies and Dipole Moments

Potential Energy Curves (PEC) for neutral hydrides for ScH, VH, MnH and CrH in various multiplicities are reported in Fig.1-4, together with available wave function theory (MCSCF+SOCI) curves. Dissociation energies for neutral hydrides in equilibrium geometry are reported in Table 1. Comparison with experimental data

[32-38], some of the published Wave Function Theory [22-24, 31] and DFT data [39] are also listed. To calculate the root mean square (rms) deviations for all theoretical values we used experimental data including the error bars. Based on rms values for bond dissociation energies, BMK gives the best agreement with experiment, followed by two WFT methods.

Table 1. Dissociation energies (kcal/mol) of neutral TM hydrides and root mean square (rms) deviations from the experimental values. Dipole moments (Debye) for neutral metal hydrides calculated with BMK and TPSS, compared with experiment and several WFT levels.

Binding Energy	**ScH**		**TiH**	**VH**		**CrH**			**MnH**		**FeH**	**CoH**	**NiH**	**CuH**	
Multiplicity	**1**	**3**	**4**	**3**	**5**	**2**	**4**	**6**	**5**	**7**	**4**	**3**	**2**	**1**	**rms**
TPSS	95.5	64.9	67.8	59.4	64.6	68.3	55.9	57.9	53.0	52.8	60.9	65.7	76.1	69.0	7.75
BMK	50.8	50.2	48.7	43.6	55.7	43.7	48.1	52.5	37.2[i]	34.8	41.0	46.4	59.5	61.8	1.38
MCSCF+SOCI[a]	47.3		43.3	36.6	42.7		37.8	44.2	21.8	33.6					1.84
MCPF[b]	51.0		47.3		53.0			48.7	21.9	39.4	45.0	44.7	64.3	61.6	1.63
Exp.	47.5 ±2.0[c]		48.9 ±2.1[d]		49.1 ±1.6[e]			44.5 ±1.6[e]		30.2 ±4.4[f]	37.5 ±1.9[g]	46.0 ±3.0[h]	59.4 ±3.0[h]	61.0 ±4.0[h]	
Dipole Moments															
BMK	1.870	2.600	2.570	2.000	2.654	2.500	1.800	3.100	1.379	1.106	2.576	2.471	3.165	2.952	
TPSS	2.530	2.800	2.616	2.100	2.497	2.900	2.600	2.905	2.053	0.756	2.630	2.566	2.304	2.468	
SDCI[j]	1.421	1.959	1.899		1.788			4.250		1.296	4.098	3.895	3.676	3.880	
CPF[j]	1.776	2.554	2.308		2.319			3.779		1.227	1.311	1.448	1.806	2.749	
MCPF[j]	1.641	2.432	2.185		2.021			3.807		1.239	2.901	2.743	2.557	2.951	
Exp.			2.455[k]					3.510[l](CrD)			2.630[m]		2.440[n]		

[a] Ref.[22-24],[b] Ref.[31],[c] Ref.[32],[d] Ref.[33],[e] Ref.[34],[f] Ref.[35],[g] Ref.[36],[h] Ref.[37-38],[i] Ref.[39],
[j] Ref.[40],[k] Ref.[41], [l] Ref.[42], [m] Ref.[43], [n] Ref.[44]

Dipole moments calculated with BMK and TPSS functionals are also reported in Table 1 and compared to the experimental data reported by Steimle et al. [41-44] and WFT results compiled by Chong et al. [40]. One can see from the Table 1, BMK and TPSS values for the dipole moments are in close agreement with experiment for TiH, FeH and CrD. For NiH the TPSS is in much better agreement with experiment, than BMK ones. For NiH and CoH the WFT results are in strong disagreement with each other, which indicated the severe difficulties in description of the electronic structure of these molecules. This clearly merits the future investigation.

4.2 Potential Energy Curves and Spin Gaps

The potential energy curves for TM hydrides are plotted on Fig. 1-4. The first two lowest multiplicities for ScH (Fig 1) are close in energy but differ in the bond length, so that singlet is more stable at the shorter, and triplet at longer bond length according to BMK results. On the contrary, TPSS overstabilizes singlet at all distances. Only singlet multiplicity is reported in the previous works [31, 9, 21, 45] including WFT study by Koseki et. al. [23]

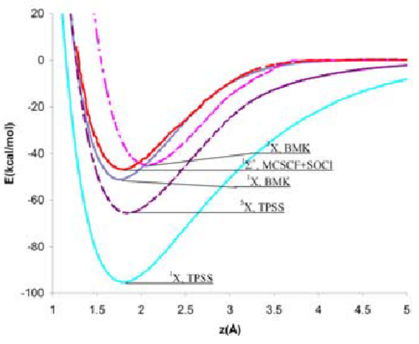


Fig. 1. Potential Energy Curves of ScH with multiplicity 1 and 3, calculated by TPSS, BMK, and WFT [23] methods

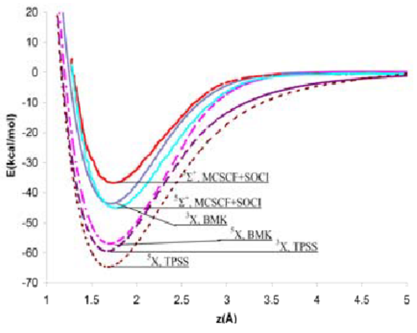


Fig. 2. Potential Energy Curves of VH with multiplicity 3 and 5, calculated by TPSS, BMK, and WFT [24] methods

Two spin multiplicities for VH are reported on Fig 2 at WFT level [24]. Both BMK and TPSS reproduce this ordering, although the spin gap in BMK is twice larger than in WFT. BMK result seems to be more reliable as it closely reproduces experimental D_e for the multiplicity 5 [34]. The two multiplicities for MnH (Fig 3) are almost degenerate in both BMK and TPSS, while WFT favors M=7 by 11 kcal/mol. Three Multiplicities of CrH are reported at WFT [3] (Fig 4), all with different dissociation limits. Both WFT and DFT predict the ground state to have the multiplicity of 6. BMK and especially TPSS underestimate the spin gap at equilibrium, as compared to the WFT predictions, while reproduce it fairly well at the dissociation limit.

The energy difference between the high and low spin states was studied previously by several authors and found to depend strongly on the fraction of HF exchange. This can be attributed to the fine balance between the negative HF exchange energy contribution from the electron of the same spin, which is opposite in sign to the electronic correlation contribution arising from the repulsion between any two electrons regardless of their spin. A method which includes the exchange and neglects the correlation (such as HF) will favor high multiplicities by maximizing the number of electrons with the same spin. On the contrary, self-interaction error in pure DFT favors low-spin states. Attempts to improve the relative spin-state energies description of density functionals include hybrid DFT schemes as well as DFT+U [46]. It was recently shown that DFT+U is capable of providing the qualitatively correct splitting in low- and high-spin iron porphyrins [46] and FeO^+ [47]. However, when hybrid DFT approach is adapted, the accurate spin energy gaps can be obtained by adjusting fraction of HF exchange in DFT functional [48]. Conradie and Ghosh [49] studied Fe(+2) spin-crossover complexes and found that pure functionals such as BLYP, PW91 and BP86 incorrectly favor spin-coupled form (covalent description), while hybrid functionals such as B3LYP lean in the other direction. To correct the latter, they suggested using B3LYP* [50-52] with reduced amount of Hartree-Fock exchange in B3LYP from the standard 20% to 15%; the B3LYP* functional has been found to

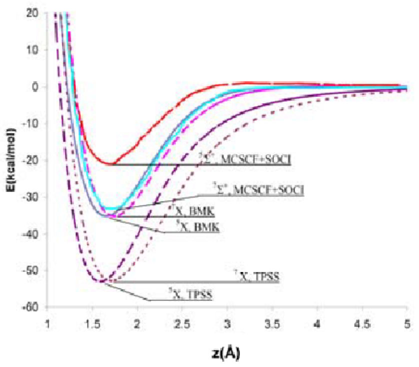


Fig. 3. Potential Energy Curves of MnH with multiplicity 5 and 7, calculated by TPSS, BMK, and WFT [22] methods

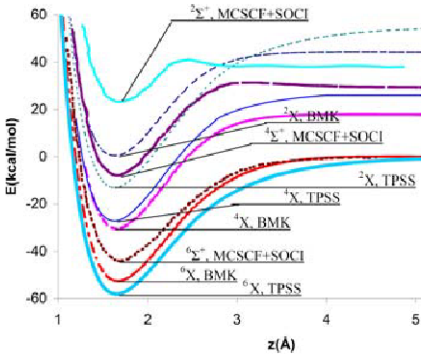


Fig. 4. Potential Energy Curves of CrH with multiplicity 2,4 and 6, calculated by non relativistic TPSS, BMK, and WFT (14) methods

give improved results. Harvey [53] also found an 15% fraction of exact exchange yields accurate results in many other cases. It appears that the large fraction of HF exchange is necessary for correct prediction of the dissociation energies, while smaller fraction is in better agreement with experimentally observed spin-gaps.

5 Conclusion

As a first step in design of the reactive bond order forcefield for TM compounds, we used two exchange-correlation functionals including explicit dependence on the kinetic energy density (τ-functionals) to study neutral hydrides formed by *3d*-transition metals (Sc-Cu). One of the functionals selected contained large fraction of Hartree-Fock exchange (BMK), and another one was a pure DFT functional (TPSS). Watchers basis sets [28], augmented with *f*-functions by Hay et. al. [29] were used. We have taken particular care to obtain the consistent SCF solution, including the stability analysis and Fermi smearing. In order to ensure the stability of Slater determinant in the entire range of interatomic distances, the potential energy curves were plotted and inspected for discontinuities. When found, the discontinuities were eliminated by using the lower energy orbitals as initial guess to continue the curve smoothly. The spin orbitals at the dissociation limit were inspected and reordered if necessary.

Qualitatively correct description of the bond dissociation was ensured by allowing the spatial and spins symmetry to break, which resulted in some spin-contamination for several systems at equilibrium. However, our calculated BMK dissociation energies are in better agreement with experiment than those obtained with high level wavefunction theory methods from the literature. The next step in this study will be the validation of BMK functional for the metal-metal and metal-carbon potential curves. These curves will be used to calibrate parameters for the reactive bond order force field and the subsequent Molecular Dynamics study of SWNT/metal growth process.

Acknowledgements

This work is supported in part by UCF start up grant. SG gratefully acknowledges Interdisciplinary Information Science Laboratory (I2lab) fellowship. The computer time was generously provided by Stokes HPC facilities at UCF Institute for Simulation and Training (IST), UCF I2lab, and DOE NERSC.

References

1. Lolli, G., et al.: Tailoring (n,m) structure of single-walled carbon nanotubes by modifying reaction conditions and the nature of the support of CoMo catalysts. J. Phys. Chem. B. 110, 2108–2115 (2006)
2. Sinnott, S.B., et al.: Model of carbon nanotube growth through chemical vapor deposition. Chem. Phys. Lett. 315, 25–30 (1999)
3. Ding, F., Rosen, A., Bolton, K.: Molecular dynamics study of the catalyst particle size dependence on carbon nanotube growth. J. Chem. Phys. 121, 2775–2779 (2004)
4. Kohn, W., et al.: Density functional theory of electronic structure. J. Phys. Chem. 100, 12974–12980 (1996)
5. Kohn, W., Sham, L.J.: Self-Consistent Equations Including Exchange and Correlation Effects. Phys. Rev. 140, 1133 (1965)
6. Geerlings, P., et al.: Conceptual density functional theory. Chem. Rev. 103, 1793–1873 (2003)
7. Pettersson, L.G.M., et al.: Positive-Ions of the 1st-Row and 2nd-Row Transition-Metal Hydrides. J. Chem. Phys. 87, 481–492 (1987)
8. Barone, V., et al.: Comparison of conventional and hybrid density functional approaches. Cationic hydrides of first-row transition metals as a case study. Chem. Phys. Lett. 249, 290–296 (1996)
9. Furche, F., Perdew, J.P.: The performance of semilocal and hybrid density functionals in 3d transition-metal chemistry. J. Chem. Phys. 124, 27 (2006)
10. Jensen, K.P.: Performance of density functionals for first row transition metal systems. J. Chem. Phys. 126, 14 (2007)
11. Bagus, P.S., et al.: Electronic-Structure of Transition-Metal Hydrides - NiH and PdH. Phys. Rev. A. 23, 461–472 (1981)
12. Blomberg, M.R.A., et al.: A Theoretical-Study of NiH - Optical-Spectrum and Potential Curves. Mol. Phys. 47, 127–143 (1982)
13. Marian, C.M., et al.: Multireference and Relativistic Effects in NiH. J. Chem. Phys. 91, 3589–3595 (1989)
14. Pouamerigo, R., et al.: The Chemical-Bonds in CuH, Cu-2, NiH, and Ni-2 Studied with Multiconfigurational 2nd-Order Perturbation-Theory. J. Chem. Phys. 101, 4893–4902 (1994)
15. Anglada, J., et al.: Theoretical Investigation of the Low-Lying Electronic States of TiH. Mol. Phys. 69, 281–303 (1990)
16. Bauschlicher, C.W.: Full Configuration-Interaction Benchmark Calculations for TiH. J. Phys. Chem. 92, 3020–3023 (1988)
17. Dai, D.G., et al.: Spectroscopic Properties and Potential-Energy Curves for 21 Electronic States of CrH. J. Mol. Spec. 161, 455–465 (1993)
18. Freindorf, M., Marian, C.M., Hess, B.A.: Theoretical-Study of the Electronic-Spectrum of the CoH Molecule. J. Chem. Phys. 99, 1215–1223 (1993)

19. Raghavachari, K., Sunil, K.K., Jordan, K.D.: Theoretical-Study of the Bonding in CuH and Cu_2. J. Chem. Phys. 83, 4633–4640 (1985)
20. Walch, S.P., Bauschlicher, C.W.: CASSCF/Cl Calculations for 1st Row Transition-Metal Hydrides - the TiH (4-Phi), VH (5-Delta), CrH (6-Sigma+), MnH (7-Sigma+), FeH (4,6-Delta), and NiH (2-Delta) States. J. Chem. Phys. 78, 4597–4605 (1983)
21. Kunz, A.B., Guse, M.P., Blint, R.J.: Potential-Energy Curves for ScH. J. Phys. B-Atom. Mol. Opt. Phys. 8, L358–L361 (1975)
22. Koseki, S., Matsushita, T., Gordon, M.S.: Dissociation potential curves of low-lying states in transition metal hydrides. 3. Hydrides of groups 6 and 7. J. Phys. Chem. A. 110, 2560–2570 (2006)
23. Koseki, S., et al.: Dissociation potential curves of low-lying states in transition metal hydrides. I. Hydrides of Group 4. J. Phys. Chem. A. 106, 785–794 (2002)
24. Koseki, S., et al.: Dissociation potential curves of low-lying states in transition metal hydrides. 2. Hydrides of groups 3 and 5. J. Phys. Chem. A. 108, 4707–4719 (2004)
25. Marti, K.H., et al.: Density matrix renormalization group calculations on relative energies of transition metal complexes and clusters. J. Chem. Phys. 128 (2008)
26. Frisch, M.J., et al.: GAUSSIAN03, Rev. D1. Gaussian Inc., Wallingford (2004)
27. Wachters, A.J.: Gaussian Basis Set for Molecular Wavefunctions Containing Third-Row Atoms. J. Chem. Phys. 52, 1033 (1970)
28. Hay, P.J.: Gaussian Basis Sets for Molecular Calculations - Representation of 3d Orbitals in Transition-Metal Atoms. J. Chem. Phys. 66, 4377–4384 (1977)
29. Harris, J.: Simplified Method for Calculating the Energy of Weakly Interacting Fragments. Phys. Rev. B. 31, 1770–1779 (1985)
30. Schaftenaar, G., Noordik, J.H.: Molden: a pre- and post-processing program for molecular and electronic structures. J. Comp.-Aid. Mol. Des. 14, 123–134 (2000)
31. Harrison, J.F.: Electronic structure of diatomic molecules composed of a first-row transition metal and main-group element (H-F). Chem. Rev. 100, 679–716 (2000)
32. Kant, A., Moon, K.A.: Mass-Spectrometric Determination of the Dissociation-Energies of Gaseous AiH, GaH, InH, ScH, and CoH and Estimation of the Maximum Dissociation-Energies of TlH, CrH, MnH, and FeH. High Temp. Science 14, 23–31 (1981)
33. Chen, Y.M., Clemmer, D.E., Armentrout, P.B.: The Gas-Phase Thermochemistry of TiH. J. Chem. Phys. 95, 1228–1233 (1991)
34. Chen, Y.M., Clemmer, D.E., Armentrout, P.B.: Gas-Phase Thermochemistry of VH and CrH. J. Chem. Phys. 98, 4929–4936 (1993)
35. Sunderlin, L.S., Armentrout, P.B.: Reactions of Mn+ with I-C4H10, Neo-C5H12 $(CH_3)_2CO$, Cyclo-C_3H_6, and Cyclo-C_2H_4O - Bond-Energies for $MnCH_2$+, MnH, and $MnCH_3$. J. Phys. Chem. 94, 3589–3597 (1990)
36. Schultz, R.H., Armentrout, P.B.: The Gas-Phase Thermochemistry of FeH. J. Chem. Phys. 94, 2262–2268 (1991)
37. Kant, A., Moon, K.A.: Dissociation-Energies of Gaseous CuH, AgH, AuH, and NiH. High Temp. Science. 11, 52–62 (1979)
38. Fisher, E.R., Armentrout, P.B.: Reactions of Co+, Ni+, and Cu+ with Cyclopropane and Ethylene-Oxide - Metal Methylidene Ion Bond-Energies. J. Phys. Chem. 94, 1674–1683 (1990)
39. Goel, S., Masunov, A.E.: Potential energy curves and electronic structure of 3d transition metal hydrides and their cations. J. Chem. Phys. 129, 214302–214314 (2008)
40. Chong, D.P., Langhoff, S.R., Bauschlicher, C.W., Walch, S.P.: Theoretical Dipole-Moments for the 1st-Row Transition-Metal Hydrides. J. Chem. Phys. 85, 2850–2860 (1986)

41. Chen, J., Steimle, T.C.: A molecular beam optical Stark study of nickel monohydride, NiH. Chem. Phys. Lett. 457, 23–25 (2008)
42. Steimle, T.C., Shirley, J.E., Simard, B., Vasseur, M., Hackett, P.: A laser spectroscopic study of gas-phase TiH. J. Chem. Phys. 95, 7179 (1991)
43. Chen, J., Steimle, T.C., Merer, A.J.: The permanent electric dipole moment of chromium monodeuteride, CrD. J. Chem. Phys. 127, 204307 (2007)
44. Steimle, T.C., Chen, J., Harrison, J., Brown, J.M.: A molecular-beam optical Stark study of lines in the (1,0) band of the $F^4\Delta_{7/2}$-$X^4\Delta_{7/2}$ transition of iron monohydride, FeH. J. Chem. Phys. 124, 184307 (2006)
45. Barone, V., Adamo, C.: First-row transition-metal hydrides: A challenging playground for new theoretical approaches. Int. J. Quan. Chem. 61, 443–451 (1997)
46. Scherlis, D.A., et al.: Simulation of heme using DFT+U: A step toward accurate spin-state energetics. J. Phys. Chem. B. 111, 7384–7391 (2007)
47. Kulik, H.J., et al.: Density functional theory in transition-metal chemistry: A self-consistent Hubbard U approach. Phys. Rev. Lett. 97 (2006)
48. Sorkin, A., Iron, M.A., Truhlar, D.G.: Density functional theory in transition-metal chemistry: Relative energies of low-lying states of iron compounds and the effect of spatial symmetry breaking. J. Chem. Theo. Comp. 4, 307–315 (2008)
49. Conradie, J., Ghosh, A.: DFT calculations on the spin-crossover complex Fe(salen)(NO): a quest for the best functional. J. Phys. Chem. B. 111, 12621–12624 (2007)
50. Reiher, M.: Theoretical study of the Fe(phen)(2)(NCS)(2) spin-crossover complex with reparametrized density functionals. Inor. Chem. 41, 6928–6935 (2002)
51. Salomon, O., Reiher, M., Hess, B.A.: Assertion and validation of the performance of the B3LYP* functional for the first transition metal row and the G2 test set. J. Chem. Phys. 117, 4729–4737 (2002)
52. Reiher, M., Salomon, O., Hess, B.A.: Reparameterization of hybrid functionals based on energy differences of states of different multiplicity. Theo. Chem. Acc. 107, 48–55 (2001)
53. Harvey, J.N.: DFT computation of relative spin-state energetics of transition metal compounds. In: Principles and Applications of Density Functional Theory, Inorg. Chem., pp. 151–183 (2004)

Darwin Approximation to Maxwell's Equations

Nengsheng Fang[1], Caixiu Liao[2], and Lung-An Ying[3]

[1] School of Mathematical Sciences, Xiamen University, Fujian, P.R.China, 361005
reyfang@gmail.com

[2] School of Mathematical Sciences, South China Normal University, Guangzhou, P.R.China, 510631
liaocaixiu800919@163.com

[3] School of Mathematical Sciences, Peking University, Beijing, P.R.China, 100080
yingla@pku.edu.cn

Abstract. Maxwell's equations are the foundation of electromagnetics, its extensive applications make computational electromagnetics a new rapidly growing subject. However, since the electric and the magnetic fields are coupling in the Maxwell system, the numerical solution of the full system of Maxwell's equations is extremely expensive in terms of computer time. Darwin model, which decouple the coupled system, is such a good approximation to Maxwell's equations. In three dimensional space case, it neglects the transverse component of the displacement current; while in two dimensional space case, Darwin model is equivalent to TE, TM models. For these reasons, it is worthwhile to investigate Darwin model, and a lot of works have been done to it. In this paper, we review the advance in the Darwin model and introduce some of our new results about the exterior problems of Darwin model and its numerical solutions with infinite elements methods.

Keywords: Maxwell's equations, Darwin model, infinite element method, exterior problem.

1 Introduction

More and more modeling and simulation of electronic devices are related to the computation of electromagnetics which becomes a very important branch in the present computational science. However, since the multiphysics fields of electric and magnetic fields are coupling together in the Maxwell's equations, there are many challenges in dealing with the coupling system directly, and the numerical solutions may be very expensive in terms of the computational cost. On the other hand, for some problems, such as the simulations of charged particle beams when no high frequency phenomenon or no rapid current charge occurs, Darwin model, which appears as a correction of the quasi-electrostatic model, including the electric fields generated by magnetic induction, is a good approximation to Maxwell's equations. This model is obtained by neglecting the solenoidal part of the displacement current in the Maxwell's equations, and exhibits an elliptic character with infinite propagation speed. Thus, the multiphysics fields of electricity and magnetics are separated in several single physical field systems which are easy to solve.

It has been proved in 1992 by P.Degond and P.A.Raviart [1] that Darwin model in three dimensional cases approximates Maxwell's equations up to the second order of the dimensionless parameter $\eta = \bar{v}/c$ for magnetic field $\mathbf{B}$ and to the third order of η for electric field $\mathbf{E}$, provided that η is small, where $\bar{v}$ is characteristic velocity and c is light velocity. While in two dimension cases, Darwin model is equivalent to TE, TM models without any approximation [15].

Some earlier investigations have been done by D.W.Hewett and C.W.Nielson etc, please refer to [2,3,4] for a derivation of the Darwin model from the plasma point and its numerical implementation by streamline methods in two dimensional bounded domains.

In 1995, P.Ciarlet and J.Zou [5] have studied the $\mathbf{H}(\mathbf{curl}; \Omega)$ and $\mathbf{H}(\mathbf{curl}, div; \Omega)$ variational formulations for the Darwin model in three dimensional bounded domains, and proved the well-posedness of the variational systems.

Since quite a number of problems in applications are related to unbounded domains, L.Ying and F.Li [6] in 2002 considered Darwin model of electric field with boundary value conditions in two dimensional exterior domains.

Recently, the Darwin model of magnetic field has been considered by C.Liao and L.Ying [7], the results are similar. An analysis of the Darwin model of approximation to Maxwell's equations in 3-D unbounded domains was given in[16].

At the same time, three dimensional exterior problem of Darwin model of both electric and magnetic fields and the numerical computation has been studied by N.Fang and L.Ying [8].

The following contents are organized as: In Section 2 we recall the relationship of Maxwell's equations and Darwin model. While in Section 3 we review the work in the foretime and introduce our recent results. And a short conclusion is given in Section 4.

2 Maxwell's Equations and Darwin Model

Maxwell's equations in vacuum are of the following form in the space-time domain $\Omega \times (0,T)$, where Ω is an open simply connected domain Ω^c in $\mathbb{R}^3$ with Lipschitz-continuous boundary $\partial\Omega$. (All the variables and function spaces in bold are denoted by vector ones):

$$\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} - \nabla \times \mathbf{B} = -\mu_0 \mathbf{J}, \quad (\textit{Ampère law}) \tag{1}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} = \mathbf{0}, \quad (\textit{Faraday law}) \tag{2}$$

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \quad (\textit{Gauss' law}) \tag{3}$$

$$\nabla \cdot \mathbf{B} = 0. \quad (\textit{no source assumption of magnetic}) \tag{4}$$

Where $\mathbf{E} = \mathbf{E}(\mathbf{x}, t)$, $\mathbf{B} = \mathbf{B}(\mathbf{x}, t)$ denote the electric field and the magnetic field respectively, and $\rho = \rho(\mathbf{x}, t)$, $\mathbf{J} = \mathbf{J}(\mathbf{x}, t)$ are respectively the charge and current densities satisfying

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{J} = 0, \qquad (charges' \ conservation \ law). \tag{5}$$

The positive constants c, ε_0, and μ_0 are respectively the light velocity, the electric permittivity, and the magnetic permeability in vacuum, which are related by

$$\varepsilon_0 \mu_0 c^2 = 1.$$

The relationship of Darwin model and Maxwell's equations are discussed in [1] when adding some initial conditions and assuming $\partial\Omega$ a perfect conductor. Since the electric field $\mathbf{E}$ can be decomposed into a sum of some transverse component $\mathbf{E}_T$ and some longitudinal component $\mathbf{E}_L$. Substituting the decomposition into equation (1) and neglecting the term $\frac{\partial \mathbf{E}_T}{\partial t}$, one can get

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \frac{1}{c^2}\frac{\partial \mathbf{E}_L}{\partial t}. \tag{6}$$

Then the Maxwell's equations (1) − (4) will be reduced to the following Darwin model, see [1,14] for deduction details:
(i) $\mathbf{E}_L = -\nabla \Phi$, and Φ is the solution of the following Dirichlet problem:

$$-\triangle \Phi = \frac{\rho}{\varepsilon_0}, \ in \ \Omega, \tag{7}$$

$$\Phi = C(t), \ on \ \partial\Omega, \tag{8}$$

$$\Phi \rightarrow 0, \quad at \ infinity. \tag{9}$$

Where $C(t)$ is an arbitrary function only depend on time t if Ω are bounded domains. If Ω represents exterior domains, $C(t)$ is a constant independent of time t, for example $C(t) = 1$ if satisfying the boundary assumptions in [5].
(ii) $\mathbf{B}$ satisfies

$$-\triangle \mathbf{B} = \mu_0 \nabla \times \mathbf{J}, \ in \ \Omega, \tag{10}$$

$$\nabla \cdot \mathbf{B} = 0, \quad in \ \Omega, \tag{11}$$

$$\mathbf{B} \cdot \mathbf{n} = \mathbf{B}_0 \cdot \mathbf{n}, \quad on \ \partial\Omega, \tag{12}$$

$$(\nabla \times \mathbf{B}) \times \mathbf{n} = \mu_0 \mathbf{J} \times \mathbf{n}, \quad on \ \partial\Omega. \tag{13}$$

(iii) $\mathbf{E}_T$ satisfies

$$\triangle \mathbf{E}_T = \frac{\partial}{\partial t}\nabla \times \mathbf{B}, \ in \ \Omega, \tag{14}$$

$$\nabla \cdot \mathbf{E}_T = 0, \ in \ \Omega, \tag{15}$$

$$\mathbf{E}_T \times \mathbf{n} = 0, \ on \ \partial\Omega, \tag{16}$$

$$< \mathbf{E}_T \cdot \mathbf{n}, 1 >= 0, \ on \ \partial\Omega. \tag{17}$$

The coupling multiphysics fields of Maxwell's equations are reduced to three single physical field systems. $\mathbf{E}_L$ relies on problem (i), a simple Poisson equation. $\mathbf{B}$ are solved by problem (ii), then plugging it into problem (iii), one can get $\mathbf{E}_T$. All the mathematical challenges to Darwin model are how to solve the problems (ii) and (iii).

3 Darwin Model and Its Numerical Methods

Both problem (ii) and problem (iii) contain equation $\nabla \cdot \mathbf{u} = 0$, i.e. (11) and (15), and it is very difficult to deal in numerical computations. All the numerical studies of Darwin model to problems (ii) and (iii) in fact were carried on to this point and gave their own ways to manage it.

D.W.Hewett and J.K.Boyd analyzed the solutions of Darwin model by streamline method in [3], and they discussed methods for calculating the magnetic field without formal vector decomposition and offered a new procedure for finding the inductive electric field. As a consequence, the numerical efforts required for each of the field time-steps are reduced, and more importantly, the needs to specify several nonintuitive boundary conditions are eliminated.

In P.Degond and P.A.Raviart's paper [1], they studied appropriate decompositions of vector fields which give rise to the well-posedness of the Darwin model. Besides, they obtained a by-product, that is the uniqueness of the variational formulations of the elliptic boundary value problem of Darwin model.

Later works are processed to tackle Darwin model through mixed variational methods by using Babuska and Brezzi's saddle point theory [12], which is said:

Let $\boldsymbol{X}$ and Q be two Hilbert spaces with norms $\|\cdot\|_{\boldsymbol{X}}$ and $\|\cdot\|_Q$ respectively, $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$ be two continuous bilinear forms defined respectively on $\boldsymbol{X}\times\boldsymbol{X}$ and $\boldsymbol{X}\times Q$, and $f(\cdot)$ and $g(\cdot)$ be two continuous linear forms defined respectively on $\boldsymbol{X}$ and Q. Then the problem : Find $(\boldsymbol{u},p)\in(\boldsymbol{X},Q)$ such that

$$\begin{aligned} a(\boldsymbol{u},\boldsymbol{v}) + b(\boldsymbol{v},p) &= f(\boldsymbol{v}),\ \forall\ \boldsymbol{v}\in\boldsymbol{X},\\ b(\boldsymbol{u},q) &= g(q),\ \forall\ q\in Q \end{aligned} \tag{18}$$

is called a saddle point problem. Let $\boldsymbol{V}$ be a closed subspace of $\boldsymbol{X}$ defined as

$$\boldsymbol{V} = \{\boldsymbol{v}\in\boldsymbol{X};\ b(\boldsymbol{v},q)=0,\ \forall\ q\in Q\}.$$

Assume that there exist two positive constants α and β such that

$$a(\boldsymbol{v},\boldsymbol{v}) \geq \alpha\|\boldsymbol{v}\|_X^2,\ \forall\ \boldsymbol{v}\in\boldsymbol{V}, \qquad (\mathbf{V}\text{-ellipticity}) \tag{19}$$

and

$$\sup_{\boldsymbol{v}\in\boldsymbol{X}} \frac{b(\boldsymbol{v},q)}{\|\boldsymbol{v}\|_{\boldsymbol{X}}} \geq \beta\|q\|_Q,\ \forall\ q\in Q. \qquad (\text{inf-sup condition}) \tag{20}$$

Then there exists a unique solution to the saddle point problem (18).

So the key point turns to investigate the mixed variational formulations of the Darwin model and to verify the **V**-ellipticity and the inf-sup condition.

The details to discuss the variational formulations of Darwin model were given by P.Ciarlet,JR and J.Zou in [5]. They have studied the $\mathbf{H}(\mathbf{curl};\Omega)$ and $\mathbf{H}(\mathbf{curl},div;\Omega)$ variational formulations for Darwin model in three dimensions, and proved the well-posedness. Nedelec's finite element method is used to solve the $\mathbf{H}(\mathbf{curl};\Omega)$ variational formulation where $\nabla\cdot\mathbf{u}=0$ is naturally satisfied in such finite element method. Moreover, they discussed a standard finite element method of so called Hood-Taylor finite element [12] for the $\mathbf{H}(\mathbf{curl}, div; \Omega)$ variational formulation. Convergence and the error estimates were obtained without requiring the physical domains to be convex. They did not give numerical experiments.

Since quite a number of problems in applications are related to exterior domains, L.Ying and F.Li in 2002 considered the exterior problem of Darwin model in [6]. They first considered Darwin model of electric field with boundary value conditions in two dimensional exterior domains. In order to show the coercivity in weighted space $\mathbf{W}$ whose elements belong to $\mathbf{H}(\mathbf{curl}, div; \Omega)$ in finite domain while in the infinity distance belong to weighted space $\mathbf{H}^{1,*}(\Omega)$ (see [9] for a definition), they spent a lot of energy and at last made clear that

$$\|\mathbf{u}\|_* = \{\|\nabla \times \mathbf{u}\|_0^2 + \|\nabla \cdot \mathbf{u}\|_0^2 + < \mathbf{u} \cdot \mathbf{n}, 1 >_{\partial\Omega}^2\}^{\frac{1}{2}}$$

is only a semi-norm on $\mathbf{W}$. With analytic function theory, they got a null space $\mathbf{V}_0$ of two degrees of freedom. Then they were able to establish the variational formulation and proved its well-posedness in the quotient space $\mathbf{W}/\mathbf{V}_0$. Numerical examples by infinite element methods were given as well, and $P_2 - P_0$ elements and square exterior domain with triangle meshes were chosen to implement the calculations. They got a so-called transfer matrix [10] in virtue of Stoke's equations since the solution of Darwin model was just that of Stoke's equation with appropriate boundary value and inhomogeneous terms. Numerical results illustrated three-order convergence under L^2-norm and one-order convergence under $\|\cdot\|_*$ norm, which was quite good in accordance with the theoretical error estimate results.

Later on, the magnetic field of Darwin model has been considered by C.Liao and L.Ying [7]. The results are similar, and the difference is that there are three degrees of freedom in the null space because the boundary conditions are different. Although differing from a function in $\mathbf{V}_0$, they manage to prove the uniqueness of the solutions if imposing a boundary condition at the infinity $\mathbf{u}|_{|x|\to\infty} = 0$ for Darwin model. This was once left as an open problem in [6].

Meanwhile, the three dimensional exterior problem of Darwin model and its numerical computation has been studied by N.Fang and L.Ying [8]. Quite different from the two dimensional exterior domains cases, $\|\cdot\|_*$ is just a norm in three dimensional case, such fact was proved through potential theory. So the well-posedness of the variational formulations were able to proved directly in $\mathbf{W}$. Moreover, for the need of investigating the Darwin model, they proved an important decomposition that any function in exterior domain Ω belonging to $\mathbf{L}^2(\Omega)$ can be decomposed to the sum of a gradient function and a rotational function, i.e. *For any $\boldsymbol{f} \in \boldsymbol{L}^2(\Omega)$, it has the decomposition $\boldsymbol{f} = \nabla\phi + \nabla \times \boldsymbol{u}$, where $\phi \in H_c = \{\phi \in H^{1,*}(\Omega); \nabla\phi \times \boldsymbol{n} = 0\}$, $\boldsymbol{u} \in \boldsymbol{B}_0$* (see [8] for the definitions). Numerical experiments with infinite element method are carried on for axisymmetric fields. The difficulty is that the variational formulation has to be considered in some weighted Sobolev spaces, the results show the convergence once again.

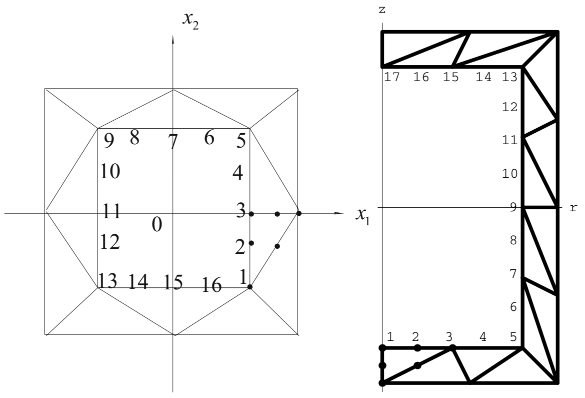


Fig. 1. Sketch maps of first layer grids in two spacial dimensions and three dimensional axisymmetric space cases

The infinite element method, which is suitable particularly for unbounded domain problems and singular problems, has its complete academic foundations and wide applications [10]. One great character of the infinite element method distinguished it with other methods is to design a similar triangulation with infinite grids for the question domain. The similar triangulation, which divides the interested domain into layers one by one with a proportionality, is the same as that of classical finite element methods in each layer. The outstanding point is that the infinite element method can approximate an exterior problem to the infinite or a singular problem to the singular position. Of course, it will get a discrete system with infinite algebraic equations. However, because in each layer the stiffness matrices are similar in a proportion, there exists a real transfer matrix which can transfer the values from one layer to the other layers. Thus, the whole computational workload for the infinite element method is reduced to solve a classical finite element in the first layer and to find the transfer matrix. For many exterior or singular problems, such as elliptic problems and singular problems, it is economical and efficient to use the infinite element method to find the solutions.

The following are parts of the numerical results of Darwin model in two spacial dimensions [6,7] and three dimensional axisymmetric space cases with infinite element methods [8]. Fig.1 are sketch maps of the first layer grids of the infinite element triangulations in two spacial dimension and three dimensional axisymmetric domains.

Example 1. (Tested by F.Li [6]) Two dimensional Darwin model of electric field with exact solutions

$$\mathbf{u} = (\frac{cos\,\theta}{r}, \frac{-sin\,\theta}{r}).$$

It calculates the errors on the (m+1) layers, where N is the number of nodes on Γ_0. One can find in Table 1 more than two order and nearly one order convergence under corresponding norms which obeys the theoretical results.

Table 1. $\xi = 1.20$, m=20, errors of the electricity field Darwin model

N	$\| \mathbf{u}_1 - \mathbf{u}_{1h} \|_0$	order	$\| \mathbf{u}_2 - \mathbf{u}_{2h} \|_0$	order	$\| \mathbf{u} - \mathbf{u}_h \|_*$	order
16	7.097883E-02		7.104025E-02		0.705941	
32	9.723456E-03	2.8678	9.723475E-03	2.8691	0.385342	0.8734
64	1.591538E-03	2.6110	1.142021E-03	3.0899	0.196713	0.9700
128	1.341634E-04	3.5684	1.491097E-04	2.9371	9.862373E-02	0.9961

Example 2. (Tested by C.Liao [7]) Two dimensional Darwin model of magnetic field with exact solutions

$$u_1 = \frac{x^3 - 3xy^2}{(x^3 - 3xy^2)^2 + (3x^2y - y^3)^2}, \quad u_2 = \frac{3x^2y - y^3}{(x^3 - 3xy^2)^2 + (3x^2y - y^3)^2},$$

the numerical results are similar as Example 1, see Table 2.

Table 2. $\xi = 1.10$, m=20, errors of the magnetic field Darwin model

N	$\| \mathbf{u}_1 - \mathbf{u}_{1h} \|_0$	order	$\| \mathbf{u}_2 - \mathbf{u}_{2h} \|_0$	order	$\| \mathbf{u} - \mathbf{u}_h \|_*$	order
16	0.203937		0.205719		1.403172	
32	2.987612E-02	2.7712	3.101684E-02	2.7696	0.543875	1.3674
64	4.002932E-03	2.8999	4.170768E-03	2.8547	0.180869	1.5884

Example 3. (Tested by N.Fang [8]) Three dimensional axisymmetric cases, an exact solution expressed in cylindrical coordinate is

$$\tilde{\mathbf{u}} = (\tilde{u}, \tilde{v}) = (-\frac{3}{16}\frac{rz}{(r^2+z^2)^{5/2}}, \frac{1}{16}\frac{r^2 - 2z^2}{(r^2+z^2)^{5/2}}). \tag{21}$$

Table 3. $\xi = 1.20$, m=20, errors of the electric field Darwin model

N	ξ	$\| \tilde{\mathbf{E}} - \tilde{\mathbf{E}}_h \|_{L_1^2}$	order	$\| \tilde{\mathbf{E}} - \tilde{\mathbf{E}}_h \|_\dagger$	order
17	1.200	0.121738		1.087437	
33	1.100	0.024777	2.29669	0.457862	1.24794
65	1.050	6.38095E-03	1.95716	0.226430	1.01585
129	1.025	1.84555E-03	1.78972	0.125478	0.85163

Table 4. $\xi = 1.20$, m=20, errors of the magnetic field Darwin model

N	ξ	$\| \tilde{\mathbf{B}} - \tilde{\mathbf{B}}_h \|_{L_1^2}$	order	$\| \tilde{\mathbf{B}} - \tilde{\mathbf{B}}_h \|_\dagger$	order
17	1.200	0.512389		2.384971	
33	1.100	0.118179	2.11627	1.081343	1.14115
65	1.050	3.03803E-02	1.93981	0.520884	1.05379
129	1.025	8.47473E-03	1.84189	0.267462	0.96163

Table 5. Errors with different $\xi's$ with $N = 33$ in domain S_2

ξ	$\| \tilde{\mathbf{E}} - \tilde{\mathbf{E}}_h \|_{L^2_1}$	$\| \tilde{\mathbf{E}} - \tilde{\mathbf{E}}_h \|_\dagger$	$\| \tilde{\mathbf{B}} - \tilde{\mathbf{B}}_h \|_{L^2_1}$	$\| \tilde{\mathbf{B}} - \tilde{\mathbf{B}}_h \|_\dagger$
2.60	0.989638	3.374672	4.513487	7.374385
1.85	0.330322	2.264038	2.163367	5.536487
1.54	0.120657	1.743531	0.058450	4.115571
1.20	0.024218	1.087437	0.014065	2.384973
1.05	0.027387	0.845684	0.011470	1.314983
1.01	0.020547	0.837463	0.009477	1.237863

Table 6. The solved components of electric field on Γ_0 with $N = 33$ and different $\xi's$

nodes	$\xi = 1.80$	$\xi = 1.50$	$\xi = 1.20$	$\xi = 1.10$	$\xi = 1.01$	exact values
1	-0.1210545	-0.1257055	-0.1260910*	-0.1260891	-0.1215592	-0.1250000
2	-0.1126088	-0.1176541	-0.1190080	-0.1186574	-0.1145254	-0.1193081
3	-0.0975158	-0.1024953	-0.1040261	-0.1037537	-0.9970447	-0.1040640
4	-0.0797613	-0.0829016	-0.0837716	-0.0827922	-0.0791954	-0.0836353
5	-0.0595527	-0.0615719	-0.0626015	-0.0618062	-0.0603292	-0.0626099
6	-0.0429919	-0.0439044	-0.0443304	-0.0437183	-0.4435603	-0.0441075
7	-0.0281114	-0.0286289	-0.0294198	-0.0290186	-0.0297962	-0.0294402
8	-0.0186065	-0.0186243	-0.0187584	-0.0184425	-0.0189317	-0.0186243
10	0.0389879	0.0396177	0.0397538	0.0396909	0.0399477	0.0396062
11	0.0444103	0.0457516	0.0461985	0.0462046	0.0464945	0.0460800
12	0.0480863	0.0503425	0.0512331	0.0513768	0.0512298	0.0513873
13	0.0497539	0.0527489	0.0539654	0.0542693	0.0535041	0.0536656
14	0.0451329	0.0484963	0.0500021	0.0505106	0.0472572	0.0506029
15	0.0363879	0.0395276	0.0407025	0.0410631	0.0272850	0.0402827
16	0.0194135	0.0214026	0.0221285	0.0225052	0.0036119	0.0225464
17	-0.0001567	0.0002899	0.0001114	0.0003259	0.0014535	0.0000000
18	-0.0204573	-0.2158120	-0.0222988	-0.0225053	-0.0006429*	-0.0225464
19	-0.0372877	-0.0394444	-0.0404561	-0.0403204	-0.0294834	-0.0402827
20	-0.0466633	-0.0492025	-0.0503285	-0.0502459	-0.049193	-0.0506029
21	-0.0505969	-0.0528761	-0.0537361	-0.0537916	-0.0563653	-0.0536656
22	-0.0487024	-0.0505964	-0.0513363	-0.0514396	-0.0524078	-0.0513873
23	-0.0442433	-0.0454351	-0.0460040	-0.0463146	-0.0465934	-0.0460800
24	-0.0387989	-0.0394013	-0.0396301	-0.0397049	-0.039857	-0.0396062
26	-0.0190566	-0.0192702	-0.0190187	-0.0188858	-0.0157653	-0.0186243
27	-0.0283864	-0.0291424	-0.0291973	-0.0292054	-0.0254894	-0.0294402
28	-0.0424493	-0.0442339	-0.0444911	-0.4438434	-0.0412477	-0.0441075
29	-0.0576513	-0.0609854	-0.0622284	-0.0623327	-0.0616994	-0.0626099
30	-0.0758385	-0.0813233	-0.0834862	-0.0835264	-0.0857989	-0.0836353
31	-0.0920192	-0.0999494	-0.1037470	-0.1041031	-0.1100688	-0.1040640
32	-0.1040798	-0.1137892*	-0.1184702	-0.1188476	-0.1290048	-0.1193081
33	-0.1094091*	-0.1204418	-0.1260022	-0.1264943*	-0.1389577	-0.1250000
Max error*	0.12473	4.4154E-02	1.1926E-02	1.1950E-02	0.17522	

Table 3 and Table 4 show that about first order convergence under the norms $\| \cdot \|_\dagger$ for both the magnetic and the electric field are obtained. Table 5 lists the errors in the domain $S_2 = \{\mathbf{x}; 1 \leq |r|, |z| \leq 32\}$, and the error tends a level when ξ decrease. Here ξ is the proportionality constant of infinite element triangulation, and m is the numbers of layers been calculated.

Example 4. (Tested by N.Fang [8]) Comparisons of local quantities of interest are made in Table 6 for the axisymmetric electric field case. Axisymmetric magnetic field and two dimensional space cases are similar [6,7,8]. The numerical experiments illustrate that one can find "good" solutions by refining the meshes and decrease the proportion ξ at the same time.

4 Conclusions

It gives at least a way to deal with some multiphysics models by decomposing the coupling system. Darwin model is such a successful example which approximates Maxwell's equations well enough. Hereto, Darwin models are thoroughly studied both in bounded domains and exterior domains especially by variational methods. Our investigations in establishing the general theory are mostly about the exterior problems. Suitable tools like analytic functions theory, potential theory and weighted Sobolev spaces are utilized to finish the theoretical analysis. Besides, infinite element methods are effective to simulate the models in unbounded domains, this was verified by the numerical experiments. Of course, as an approximate model to Maxwell's equations, Darwin model are worthy of further study. It leaves to combine the problems of (i), (ii) and (iii) together. Some related works such as adaptive finite elements to Darwin model are being studied by the authors.

References

1. Degond, P., Raviart, P.A.: An analysis of the Darwin model of approximation to Maxwell's equations. Forum Math. 4, 13–44 (1992)
2. Hewett, D.W., Nielson, C.W.: A multidimensional quasineutral plasma simulation model. J. Comput. Phys. 29, 219–236 (1978)
3. Hewett, D.W., Boyd, J.K.: Streamlined Darwin simulation of nonneutral plasmas. J. Comput. Phys. 70, 166–181 (1987)
4. Nielson, C.W., Lewis, H.R.: Particle code models in the non radiative limit. Methods Comput. Phys. 16, 367–388 (1976)
5. Ciarlet Jr., P., Zou, J.: Finite element convergence for the Darwin model to Maxwell's equations. Math. Modeling Numer. Anal. 31, 213–250 (1997)
6. Ying, L., Li, F.: Exterior problem of the Darwin model and its numerical computation. Math. Modeling Numer. Anal. 37, 515–532 (2003)
7. Liao, C., Ying, L.: Exterior problem of the magnetic field in Darwin model and its numerical computation. Numer. Method PDE 24(2), 418–434 (2008)
8. Fang, N., Ying, L.: Three dimensional exterior problem of the Darwin model and its numerical computation. Math. Model and Methods in Appl. Science 18(10), 1673–1701 (2008)

9. Ying, L.: Numerical Methods for Exterior problems. World Scientific, Singapore (2006)
10. Ying, L.: Infinite Element Methods. Peking University Press, Beijing and Vieweg and Sohn Verlagsgesellschaft mbH, Braunschweig/wiesbaden (1995)
11. Girault, V., Sequeira, A.: A well-posed problem for the exterior Stokes equations in two and three dimensions. Arch. Ration. Mech. Anal. 114, 313–333 (1991)
12. Girault, V., Raviart, P.A.: Finite Element Methods for Navier-Stokes Equations: Theory and Algorithms. Springer, Berlin (1986)
13. Bernardi, C., Raugel, G.: Analysis of some mixed finite element methods for the Stokes problem. Math. Comput. 44(169), 71–79 (1985)
14. Li, T., Qin, T.: Physics and Partial Differential Equations (in Chinese). Higher Education Press, Beijing (1997)
15. Liao, C., Ying, L.: Equivalence of TE,TM model and Darwin model (preprint)
16. Liao, C., Ying, L.: An analysis of the Darwin model of approximation to Maxwell equations in 3-D unbounded domains. Comm. Math. Sci. 6(3), 695–710 (2008)

Study of Parallel Linear Solvers for Three-Dimensional Subsurface Flow Problems

Hung V. Nguyen, Jing-Ru C. Cheng, and Robert S. Maier

U.S. Army Engineer Research and Development Center
Major Shared Resource Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199
{Hung.V.Nguyen,Ruth.C.Cheng,Robert.S.Maier}@usace.army.mil

Abstract. The performance of an iterative method for solving a system of linear equations depends on the structure of the system to be solved and on the choice of iterative solvers in combination with preconditioners. In this paper, the performance of a specified set of linear solvers and preconditioners provided by PETSc and Hypre is evaluated based on three data sets from subsurface finite element flow models. The results show that simple preconditioners are robust but do not enable convergence behavior that scales with problem size. They also show that it is important to choose an appropriate type of solver for different kind of simulations.

Keywords: Sparse parallel solvers, iterative solvers, PETSc, Hypre, and finite element.

1 Introduction

Simulations of groundwater [3] and watershed [1] flow require the solution of a nonlinear system of partial differential equations (PDE). Discretization of the PDE spatial domain on a finite-element mesh results in a set of time-dependent, nonlinear algebraic equations that may be solved by a Newton or Modified Picard algorithm. These algorithms require the solution of one or more large sparse systems of linear equations at each time-step, of the form,

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

where $\mathbf{A}=[a_{ij}]$ is an $n \times n$ matrix and $\mathbf{b}$ a given right-hand-side vector.

WASH123 is a simulation code used for modeling time-dependent surface and subsurface flows, and the coupling between the two flow regimes [2]. WASH123 uses a modified Picard algorithm to solve the nonlinear problem of three-dimensional (3-D) subsurface flow, and the resulting linear systems can be shown to be symmetric and positive definite (SPD) in all cases. Also, the matrix $\mathbf{A}$ may vary over the course of a time-dependent simulation depending on the forcing, coupling, and boundary conditions. A significant part of the computation time of a WASH123 simulation is spent solving the linear system [5], [6]. Therefore, the performance of linear solvers is of great interest.

Iterative methods are most often used for linear problems involving a large number of variables because of the memory and scalability restrictions of direct methods, such as LU decomposition. Krylov subspace methods are an important class of iterative solution solvers [4], [8]. This class includes the Conjugate Gradient (CG) method, which is robust for SPD matrices. Tracy and Gavali have previously tested a wide range of iterative solvers [4] on matrices arising from the FEMWATER code, a predecessor of WASH123. Their results suggest that CG remains a competitive method for matrices in the range of a few million unknowns, but leaves open the question of the most effective preconditioner. In practice, combining a Krylov subspace method with a preconditioner is essential, especially for an ill-conditioned linear system. Therefore, this study tests the effectiveness of iterative solvers combined with various preconditioners to determine those most effective for subsurface and coupled watershed flow applications.

The convergence criterion for an iterative method is typically tied to the residual, $r^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$, where superscript k denotes the kth iteration. The PETSc convergence criterion was used:

$$\|\mathbf{r}^k\|_2 < \varepsilon \|\mathbf{b}\|_2 \tag{2}$$

where ε is relative tolerance, $\varepsilon = 10^{-12}$ for linear system A and B; $\varepsilon = 10^{-16}$ for linear system C.

2 Numerical Libraries

Two numerical libraries were used in this study: PETSc [4] and Hypre [9]. Both provide parallel routines for solving large sparse linear systems. PETSc, the Portable, Extensible Toolkit for Scientific computation, provides a variety of preconditioners and Krylov subspace solvers. This study used Jacobi, Block Jacobi (BJacobi), successive overrelaxation (SOR), and the Additive Schwartz method (ASM) as preconditioners for CG. BJacobi and ASM were implemented with one block per processor. The individual blocks were solved with ILU(0). Hypre [9] in addition provides multigrid preconditioning, and the study also used Hypre's algebraic multigrid method, boomerAMG, as a preconditioner for CG. The study also experimented with the Support Tree Conjugate Gradient (STCG) method [7], as implemented in PETSc [4]. The STCG method is potentially more effective than CG where **A** is a generalized Laplacian [7], but relatively little is known about its performance in practical applications.

3 Test Problems

The choice of the best iterative solver may strongly depend on the grid size, the aspect ratio of the grid, and physical parameters of the problem. Typically, the convergence rate of the iterative solver is strongly connected to the condition number of the matrix, $\kappa(\mathbf{A})$. Therefore, this study used three sets of data, which try to depict the important role of these characters.

3.1 Linear System A

The FEMWATER code [3] was used to create linear system A. The data set is generated from a pump-and-treat model developed for remediation of contaminated groundwater. The 3-D mesh contains 102,996 nodes, 187,902 elements, and 8 material types, for a total of 31 layers. FEMWATER uses Picard linearization, which produces a symmetric, positive-definite (SPD) linear system. The condition number of the coefficient matrix **A,** $\kappa(\mathbf{A}) = 2.7 \times 10^6$.

3.2 Linear System B

The WASH123D watershed code was constructed based on its first-principle, physics-based mathematical model [2]. In WASH123D, the cross-section-averaged 1-D diffusive wave equation, the depth-averaged 2-D diffusive wave equation, and the 3-D Richards equation were solved with semi-Lagrangian for canal network flow and overland flow, and Galerkin finite element methods for variably saturated subsurface flow, respectively [2]. Also, the computational domain is discretized with finite element meshes; each element can be assigned with a different material type to account for heterogeneity; and each material may have its own set of physical model parameters. The 3-D subsurface flow in this coupled watershed model is governed by the Richards equation, solved by Picard linearization, but employs a modular software approach.

The 3-D mesh contains 59,409 nodes, 84,996 elements, and includes 17 material types for a total of six layers. Figure 1 shows the 3-D computational mesh of this linear system B.

Similar to the linear system A above, the matrix **B** is symmetric and positive definite, with condition number $\kappa(\mathbf{B}) = 3.9 \times 10^9$; thus this linear system is ill-conditioned.

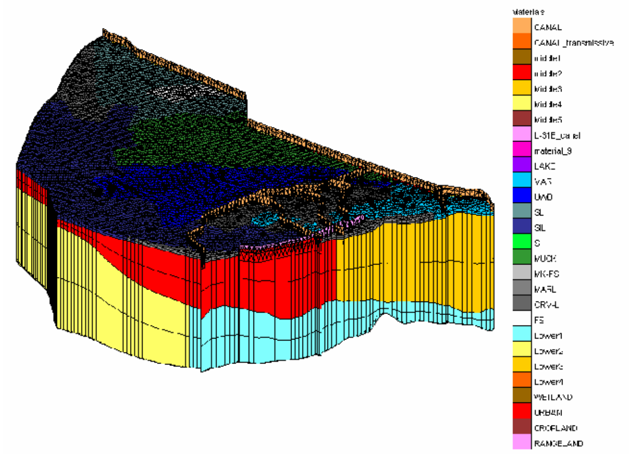


Fig. 1. The 3-D computational mesh of data set B

3.3 Linear System C

Linear system C was also generated by the WASH123 code, from the same model as linear system B, using finer mesh. The 3-D mesh for this system contains 2,124,108 nodes, 4,018,700 elements, and includes 17 material types for a total of 20 layers. The resulting matrix is symmetric and positive definite.

4 Test Results

A C code was written to read these data sets and then call the PETSc functions to create a sparse parallel matrix **A** in AIJ format, a parallel vector **b**, and independent solution **xtrue**, which was used to compare with solution from PETSc and Hypre for the purpose of verification. The code was compiled and run on a Cray XT4 containing 2,152 compute nodes; each has 2.1GHz AMD Opteron 64-bit quad-core processors and 8 GBytes of dedicated memory.

Tables 1-3 compare the convergence and performance of different preconditioners for CG and STCG methods, respectively. As expected, preconditioning can significantly increase performance of Krylov subspace solvers. Without preconditioning, the solvers often fail to converge. Both CG and STCG methods using the Jacobi preconditioner are robust solvers for all three linear systems but not necessarily the most efficient. ASM is the most effective preconditioner for linear systems B and C. The ASM preconditioner option *–pc_asm_type basic* was employed, which uses the full restriction and interpolation operator. The option *–pc_asm_type restrict* is the PETSc default, but it does not converge for linear systems A, B, or C.

Table 1. Solver time, number of iterations, and residual norm using CG (shaded) and STCG (white) methods for linear systems A (two cores)

System	**CG**			**STCG**		
PC	**Residual Norm X10^{-9}**	**Iteration**	**Time (sec)**	**Residual Norm X10^{-9}**	**Iteration**	**Time (sec)**
None	1348.94	5700	39.61	1348.94	5700	41.03
Jacobi	3.22	670	4.73	3.22	670	4.90
Bjacobi	10.20	210	3.98	3.22	210	4.05
SOR	4.08	272	5.34	4.08	272	5.43
ASM	12.59	220	5.02	12.59	220	5.08

Table 2. Solver time, number of iterations, and residual norm using CG (shaded) and STCG (white) methods for linear system B (two cores)

System	**CG**			**STCG**		
PC	**Residual Norm X10^{-9}**	**Iteration**	**Time (sec)**	**Residual Norm X10^{-9}**	**Iteration**	**Time (sec)**
None	NC	NC	NC	NC	NC	NC
Jacobi	9.05	20340	76.58	9.05	20340	78.75
Bjacobi	9.05	20340	78.11	9.05	20340	80.62
SOR	6.93	18784	173.28	6.93	18784	176.00
ASM	18.93	2818	34.66	18.93	2818	35.07

Note: NC means not converge.

Table 3. Solver time, number of iterations, and residual norm using CG (shaded) and STCG (white) methods for linear system C (two cores)

System	CG			STCG		
PC	Residual Norm X10^{-13}	Iteration	Time (sec)	Residual Norm X10^{-13}	Iteration	Time (sec)
None	NC	NC	NC	NC	NC	NC
Jacobi	1.37	105960	17123.40	1.37	105960	17836.93
Bjacobi	5.50	47517	18269.13	5.50	47517	18611.86
SOR	1.21	57391	22968.26	1.21	57391	23381.96
ASM	6.24	30784	15994.27	6.24	30784	16220.26

Note: NC means not converge.

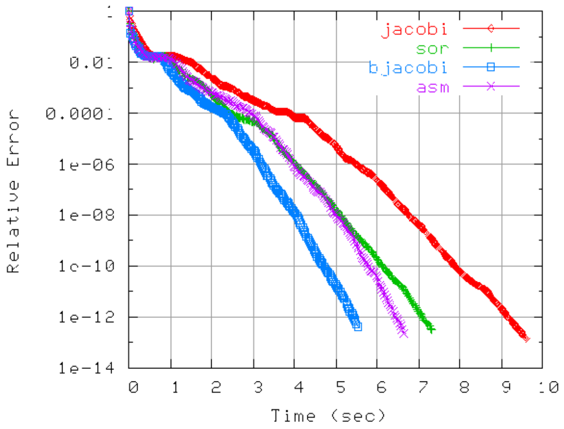


Fig. 2. Relative error versus solver time using CG method for linear system A Relative error ||**b –AX**^{k}|| / ||**b –AX**^{0}||

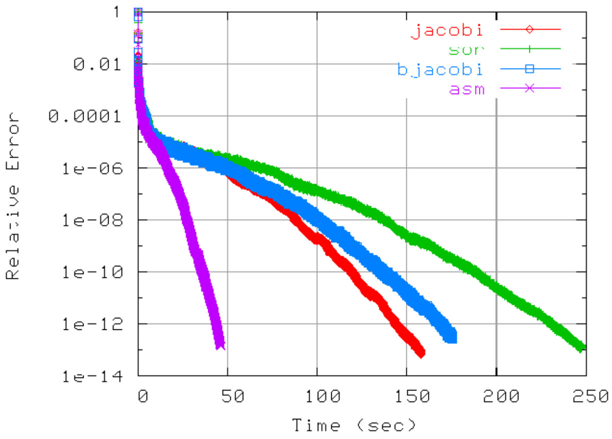


Fig. 3. Relative error versus solver time using CG method for linear system B Relative error ||**b –AX**^{k}|| / ||**b –AX**^{0}||

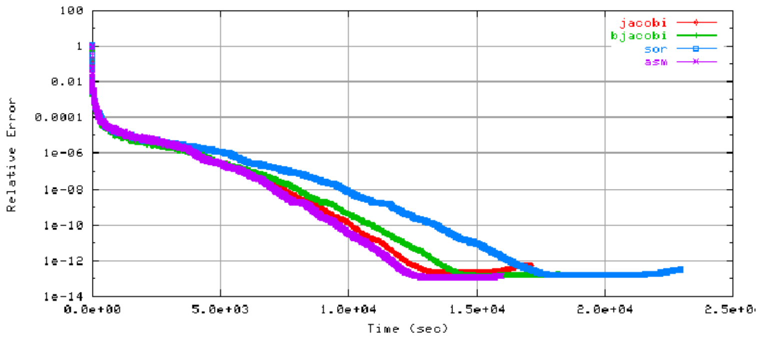


Fig. 4. Relative error versus solver time using CG method for linear system C Relative error ||**b** –**AX**^{k}|| / ||**b** –**AX**^{0}||

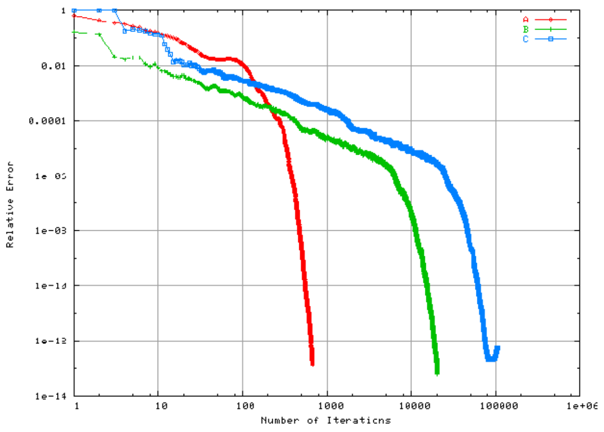


Fig. 5. Relative error versus number of iterations using CG method with Jacobi preconditioner for linear systems A, B, and C. Relative error ||**b** –**AX**^{k}|| / ||**b** –**AX**^{0}||.

Figures 2-4 compare convergence of the residual norm for different preconditioners using the CG method. For linear system A, Block-Jacobi is the most efficient in terms of solver time, followed by ASM, SOR, and Jacobi. However, for linear systems B, ASM converges in the fewest iterations and requires the least amount of time, followed by the Block-Jacobi, SOR, and Jacobi. For linear system C, ASM converges slightly faster than others, followed by Jacobi, Block-Jacobi, and SOR.

The convergence behavior of the three linear systems is compared in Figure 5 for the Jacobi preconditioner. The behavior of the three systems follows the same pattern; an initial, relatively slow rate of convergence is followed by an abrupt change to a more rapid rate of convergence. The initial stage appears to be approximately linear, or slightly faster, in the number of iterations, while in the latter stage, convergence is faster than quadratic. For these linear systems, which are closely related in terms of the underlying PDE, the initial rates of convergence are very similar, i.e., the error is reduced at the same rate for each linear system.

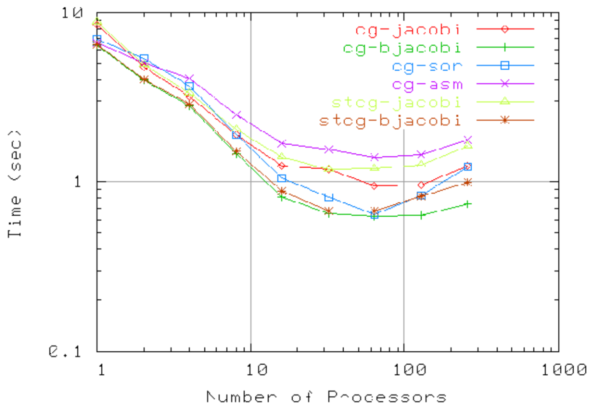


Fig. 6. Solver time using CG and STCG methods combination with Jacobi, Block Jacobi, SOR, and ASM preconditioners for linear system A

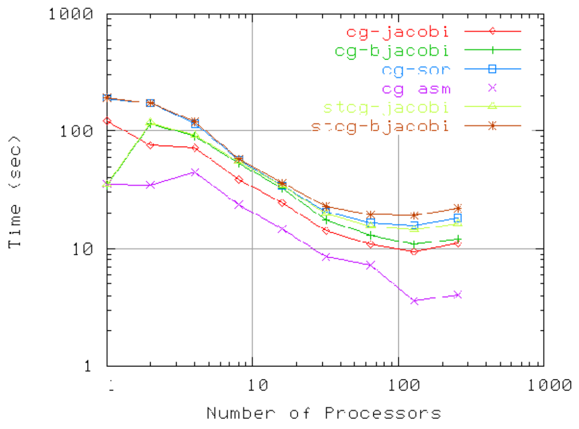


Fig. 7. Solver time using CG and STCG methods combination with Jacobi, Block Jacobi, SOR, and ASM preconditioners for linear system B

Figures 6-8 show the scaling of solver time with the number of processors using CG and STCG methods. For linear system A, the performance scales linearly up to about 16 processors and then flattens out, indicating that fewer than 5000-10,000 nodes per processor results in lower parallel efficiency. The various preconditioners have similar scaling behavior for system A. For linear system B, performance scales erratically in the range from one to four processors but scales linearly in the range from 4 to 32 processors. Beyond 32 processors, the performance flattens out for all preconditioners except ASM, which arguably demonstrates linear scaling from 4 to 128 processors. The number of blocks used in the Block-Jacobi and ASM preconditioners depends on the number of processors; because of this, their solver time running on one processor may be less than that on two, four, or eight processors.

For linear system C, performance does not scale linearly in the range from one to four processors. This problem is due to on-chip memory contention since four cores share memory in the same node on the Cray XT4 system. The performance scales linearly in the range from 4 to 256 processors.

Figure 9 shows the scaling of solver time with the number of processors using boomerAMG method from Hypre for linear systems A and B. The preliminary results are based on a residual tolerance of $\varepsilon_= 10^{-7}$.

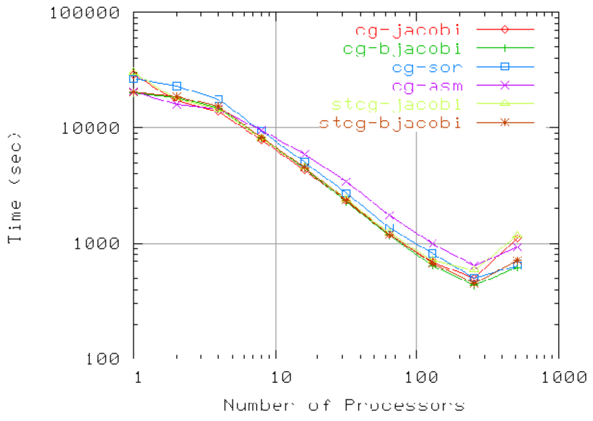


Fig. 8. Solver time using CG and STCG methods combination with Jacobi, Block Jacobi, SOR, and ASM preconditioners for linear system C

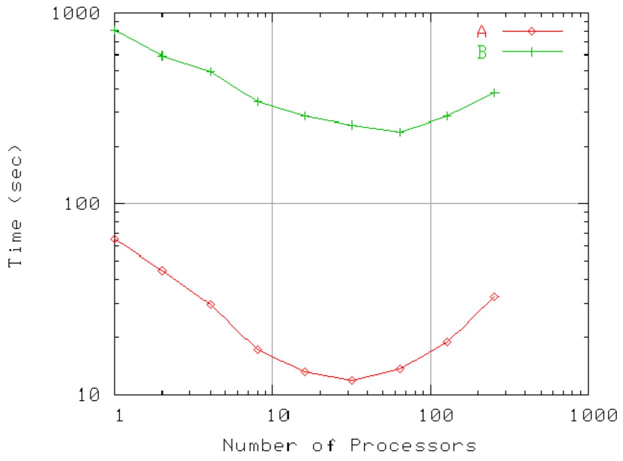


Fig. 9. Solver time using boomerAMG method for linear systems A and B

5 Summary and Future Work

Jacobi-CG appears to be the most robust method for solving moderately ill-conditioned SPD problems in the range of a few hundred-thousand unknowns. However, it is well known that the number of CG iterations will not scale linearly with matrix dimension. As WASH123 problems become larger, the number of Jacobi-CG iterations required to achieve a fixed error tolerance will grow, causing the solution time to increase faster than the number of unknowns. This study found that a multilevel preconditioner was effective for linear system B but was not effective with the same preconditioner for the closely related system C. The authors are therefore continuing to experiment with algebraic multigrid and multilevel preconditioners. Even though the cost per iteration of these preconditioners is relatively high for problems in the range of a few hundred-thousand unknowns, they may prove to be more efficient for problems with tens or hundreds of millions of unknowns, where the number of CG iterations is likely to be prohibitive. The authors also continue to investigate the convergence of CG in time-dependent problems, where a reasonable initial guess is available from the previous time-step, and convergence may be relatively fast.

Acknowledgments. This study was supported by the Department of Defense (DoD) 6.1 Basic research funding in the Civil Works category. It was also supported in part by an allocation of computer time from the DoD High Performance Computing Modernization Program.

References

1. Cheng, J.R.C., Hunter, R.M., Cheng, H.P., Richards, D.R., Yeh, G.T.: Parallelization of a watershed model|Phase III: Coupled 1-dimensional channel, 2-dimensional overland, and 3-dimensional subsurface ows. In: Computational Methods in Water Resources XVI, Copenhagen, Denmark, CMWR CD-ROM, paper 64, June 19-22 (2006)
2. Yeh, G.T., Huang, G., Zhang, F., Cheng, H.P., Lin, H.C., Cheng, J.R., Edris, E.V., Richards, D.R.: An integrated media, integrated processes watershed model WASH123D: Part 1model descriptions and features. In: Computational Methods in Water Resources XVI, Copenhagen, Denmark, CMWR CD-ROM, paper 29, June 19-22 (2006)
3. Lin, H.J., Richards, D.R., Yeh, J., Cheng, J., Cheng, H., Jones, N.L.: FEMWATER: A three-dimensional finite element computer model for simulating density dependent flow and transport, TR CHL-97-12, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS (1997)
4. Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc (2008), `http://www.mcs.anl.gov/petsc`
5. Tracy, T.F., Oppe, C.T., Gavali, S.: Testing Parallel Linear Iterative Solvers for Finite Element Groundwater Flow Problems. NAS Technical report; NAS-07-007 (September 2007)

6. Cheng, C.J.-R., Nguyen, H., Cheng, H.-P(P.), Richards, R.D.: Parallel Performance of the Coupled Watershed System with Multiple_Spatial Domains and Multiple-Temporal Scales. In: PARA2008, 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing, NTNU, Trondheim, Norway, May 13-16 (2008)
7. Gremban, D.K., Miller, L.G., Zagha, M.: PerfoMarco: Performance Evaluation of a New Parallel Preconditioner. In: IPPS 1995: Proceeding of the 9th International Symposium on Parallel Processing (1995)
8. Demmel, J.W.: Applied Numerical Linear Algebra. SIAM, Philadelphia (1997)
9. Falgout, R.D., Meier Yang, U.: hypre: A library of high performance preconditioners. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J., Hoekstra, A.G. (eds.) ICCS-ComputSci 2002. LNCS, vol. 2331, pp. 632–641. Springer, Heidelberg (2002); also available as LLNL Technical Report UCRL-JC-146175

A Domain Decomposition Based Parallel Inexact Newton's Method with Subspace Correction for Incompressible Navier-Stokes Equations*

Xiao-Chuan Cai[1] and Xuefeng Li[2]

[1] Department of Computer Science, University of Colorado at Boulder
Boulder, CO 80309
cai@cs.colorado.edu

[2] Department of Mathematics,
Loyola University New Orleans, New Orleans, LA 70118
li@loyno.edu

Abstract. There are two major types of approaches for solving the incompressible Navier-Stokes equations. One of them is the so-called projection method, in which the velocity field and the pressure field are solved separately. This method is very efficient, but is difficult to be extended to another multi-physics problem when an appropriate splitting is not available. The other approach is the fully coupled method in which the velocity and pressure fields stay together throughout the computation. The coupled approach can be easily extended to other multi-physics problems, but it requires the solution of some rather difficult linear and nonlinear algebraic systems of equations. The paper focuses on a fully coupled domain decomposition based parallel inexact Newton's method with subspace correction for incompressible Navier-Stokes equations at high Reynolds numbers. The discussion is restricted to the velocity-vorticity formulation of the Navier-Stokes equations, but the idea can be generalized to other multi-physics problems.

1 Introduction

In this paper we develop inexact Newton type methods [6] that use local subdomain corrections for large nonlinear systems of algebraic equations, arising from the discretization of nonlinear partial differential equations. All systems considered in the paper have a common feature – local high nonlinearities. In other words, the nonlinear system may have many equations, but only a small percentage of them are highly nonlinear compared to the rest of the equations. These local high nonlinearities often correspond to boundary or interior layers, or corner singularities [3,16]. Global inexact Newton's methods may be used to solve the system, but often computing time is wasted since all equations are treated equally as if they were all highly nonlinear. We introduce local zeroth- and

* The research was supported in part by DOE under DE-FC02-04ER25595, and in part by NSF under grants ACI-0305666, CNS-0420873, CCF-0634894, and CNS-0722023.

first-order Jacobi method to remove the local high nonlinearities and therefore improve the efficiency and the effectiveness of the outer global inexact Newton's method [7,15], which performs well on equations with roughly the same level of nonlinearities.

We provide numerical results to demonstrate its effectiveness as compared to the classical inexact Newton's method. As an example, we show numerically that the method performs well for solving the two-dimensional nonlinear driven cavity flow problem [8]. Using the velocity-vorticity formulation, in terms of velocity u, v, and vorticity ω, the driven cavity flow problem on unit square $\Omega = (0,1) \times (0,1)$ is

$$\begin{cases} -\Delta u - \dfrac{\partial \omega}{\partial y} & = 0 \\ -\Delta v + \dfrac{\partial \omega}{\partial x} & = 0 \\ -\dfrac{1}{Re}\Delta \omega + u\dfrac{\partial \omega}{\partial x} + v\dfrac{\partial \omega}{\partial y} & = 0. \end{cases} \tag{1}$$

Here Re is Reynolds number. The boundary conditions are:

bottom, left and right: $u = v = 0$; top: $u = 1$, $v = 0$.

The boundary condition on ω is given by its definition:

$$\omega(x,y) = -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}. \tag{2}$$

We vary the Reynolds number in the experiments. Using the usual uniform mesh finite difference approximation with the 5-point stencil (upwinding for the convective terms and central differencing for the other terms) we obtain a system of nonlinear equations in the form of

$$F(u) = 0, \tag{3}$$

where $u = (u_1, \cdots, u_N)^T$, $F = (f_1, \cdots, f_N)^T$ and $f_i = f_i(u_1, \cdots, u_N)$. Here N is the total number of unknowns. Classical Newton type algorithms, e.g. [6], do not assume that nonlinearities in functions $f_1, \cdots, f_N$ are too much different. This is often fine when the number of functions is small, but is not acceptable when N is sufficiently large. Recent experiences show that, in many large scale multi-physics problems, the nonlinearities are far from balanced. In many situations, only a small percentage of the components of F are highly nonlinear. To improve the efficiency of Newton's methods, in this paper, we develop a locally adaptive version of Newton's method which uses some special treatments for the highly nonlinear components of F.

For solving large scale problems, parallel processing is a must, and we assume here that domain decomposition [14] is used with a partition of the vector u, as well as F, into subdomains

$$u_{\Omega_1}, \cdots, u_{\Omega_p}, \text{ and } F_{\Omega_1}, \cdots, F_{\Omega_p}.$$

With this partition, the nonlinear system takes a block form as follows.

$$\begin{cases} F_1(u_{\Omega_1}, \cdots, u_{\Omega_p}) = 0 \\ F_2(u_{\Omega_1}, \cdots, u_{\Omega_p}) = 0 \\ \vdots \\ F_p(u_{\Omega_1}, \cdots, u_{\Omega_p}) = 0 \end{cases} \tag{4}$$

The classical Newton's method goes like

$$\begin{pmatrix} u^n_{\Omega_1} \\ \vdots \\ u^n_{\Omega_k} \\ \vdots \\ u^n_{\Omega_p} \end{pmatrix} = \begin{pmatrix} u^{n-1}_{\Omega_1} \\ \vdots \\ u^{n-1}_{\Omega_k} \\ \vdots \\ u^{n-1}_{\Omega_p} \end{pmatrix} + J^{-1} \begin{pmatrix} F_1(u^{n-1}_{\Omega_1}, \cdots, u^{n-1}_{\Omega_k}, \cdots, u^{n-1}_{\Omega_p}) \\ \vdots \\ F_k(u^{n-1}_{\Omega_1}, \cdots, u^{n-1}_{\Omega_k}, \cdots, u^{n-1}_{\Omega_p}) \\ \vdots \\ F_p(u^{n-1}_{\Omega_1}, \cdots, u^{n-1}_{\Omega_k}, \cdots, u^{n-1}_{\Omega_p}) \end{pmatrix}. \tag{5}$$

As one can tell from (5), all functions F_i participate in all iterations no matter where the high nonlinearities are.

On the other hand, there is another well-known method called nonlinear Jacobi method[12], and its block form can be presented as follows: Let $u^n_{\Omega_1}, \cdots, u^n_{\Omega_p}$ be the current approximate solution on each subdomain, then the new iteration on subdomain Ω_i, $u^{n+1}_{\Omega_i}$, is computed by solving

$$F_i(u^n_{\Omega_1}, \cdots, u^{n+1}_{\Omega_i}, \cdots, u^n_{\Omega_p}) = 0, \tag{6}$$

using Newton's method on Ω_i with $u^n_{\Omega_i}$ as the initial guess. These subdomain problems can be solved independently. This nonlinear Jacobi method has the perfect parallelism, and is a local method in the sense that if the output of the subdomain function on Ω_k is more nonlinear than others, then more subdomain Newton iterations are needed only on this particular subdomain. No other subdomain needs to participate in the lengthy iterations, as the global Newton's method does. However, the nonlinear Jacobi method is not being used much in practice because of its slow convergence. The focus of this paper is to combine the non-proliferation properties of Jacobi and the fast convergence of global Newton's methods.

For simplicity, we assume that the high nonlinearity is mainly in the area of Ω_k, and as a result, the subdomain residual function $\|F_{\Omega_k}\|_2$ is larger than residuals in other subdomains, i.e.,

$$\|F_{\Omega_k}\|_2 \geq \|F_{\Omega_i}\|_2, \quad i \neq k.$$

In other words, the subsolution in the k^{th} subdomain $u^n_{\Omega_k}$ is not as good as subsolutions in other subdomains. To improve the situation in subdomain Ω_k, or say to balance the overall nonlinearity, we could stop the global Newton iteration temporarily and focus on subdomain Ω_k using the following subdomain Newton iterations:

$$u^{n,m}_{\Omega_k} = u^{n,m-1}_{\Omega_k} - J^{-1}_{\Omega_k} F_{\Omega_k}(u^n_{\Omega_1}, \cdots, u^{n,m-1}_{\Omega_k}, \cdots, u^n_{\Omega_p}), \tag{7}$$

with the initial guess $u_{\Omega_k}^{n,0} = u_{\Omega_k}^n$. Suppose Algorithm (7) converges, then the subdomain Newton iterations will result in

$$\lim_{m\to+\infty} u_{\Omega_k}^{n,m} = v_{\Omega_k}^n,$$

that is

$$v_{\Omega_k}^n = v_{\Omega_k}^n - J_{\Omega_k}^{-1} F_{\Omega_k}(u_{\Omega_1}^n, \cdots, v_{\Omega_k}^n, \cdots, u_{\Omega_p}^n),$$

which implies that

$$F_{\Omega_k}(u_{\Omega_1}^n, \cdots, v_{\Omega_k}^n, \cdots, u_{\Omega_p}^n) = 0. \tag{8}$$

Now the question is whether the locally updated solution $(u_{\Omega_1}^n, \cdots, v_{\Omega_k}^n, \cdots, u_{\Omega_p}^n)$ is better than the non-updated solution $(u_{\Omega_1}^n, \cdots, u_{\Omega_k}^n, \cdots, u_{\Omega_p}^n)$ according to the overall residual function as defined in (3).

This algorithm can be extended for other multi-physics problems, such as the semiconductor device simulation problem in [11], the fluid-structure interaction problem in [2], and the magnetic reconnection problem in [13]. However, we'll focus on the incompressible Navier-Stokes equations in this paper for now. And the rest of the paper is organized as follows. In Section 2, we introduce two parallel inexact Newton's methods with subspace correction. These algorithms are applicable for general nonlinear problems with local high nonlinearities. In Section 3, we provide some numerical results for solving the incompressible Navier-Stokes equations.

2 Subdomain Jacobi–Newton Methods

Suppose $\tilde{u}^n$ is the current approximate solution from the global Newton's method. For many PDE problems with local high nonlinearities, the surface plot of $F(\tilde{u}^n)$ would show a peak, say in Ω_k, which is way higher than its neighboring regions. Many more Newton iterations are needed to remove this peak. During these Newton iterations, the components of $\tilde{u}^n$ on subdomains $\Omega_1, \cdots, \Omega_{k-1}, \Omega_{k+1}, \cdots, \Omega_p$ do not change much at all. In other words, the calculation on these subdomains is just a waste of time.

In order to isolate and to remove the peak, it seems to be a good idea not to use the original system (4), but to use the following nonlinear system of equations

$$G^n(u) = \left(G_1^n(u), \cdots, G_p^n(u)\right)^T = 0$$

defined as:

$$\begin{cases} G_k^n(u) = F_k(u) & = 0 \\ \\ G_i^n(u) = F_i(u) - F_i(\tilde{u}^n) = 0, & i \neq k. \end{cases} \tag{9}$$

We can clearly see that $G^n(u) = 0$ amounts to smoothing the worst nonlinearity in the k^{th} subdomain while maintaining others. That is, this approach eliminates the worst nonlinearity without *over-correcting* the rest of the components.

The only issue is that the new nonlinear system (9) is as expensive to solve as the original system (4). In the rest of the paper, we propose two approximations to (9). Based on these approximate peak removing algorithms, we introduce the so-called Subdomain Jacobi–Newton (SJN) methods.

Algorithm 1 (Subdomain Jacobi–Newton Framework). *For* $n=1,2,\cdots,$

1. *Perform the classical Newton iteration and generate a tentative iterate* $\tilde{u}^n$*.*
2. *Convergence test. If* $\|F(\tilde{u}^n)\| \leq \epsilon_{\mathrm{r}}\|F(u^0)\|$ *or* $\|F(\tilde{u}^n)\| \leq \epsilon_{\mathrm{a}}$*, a solution is found, or else go to next step. Here,* ϵ_{r} *and* ϵ_{a} *represent the relative and absolute machine epsilons, respectively, for the computation.*
3. *Inner iteration.*
   - (a) *Peak-finding. Find* k *such that* $\|F_k(\tilde{u}^n)\|_2 \geq \|F_i(\tilde{u}^n)\|_2$*,* $\forall i \neq k$ *and* $\|F_k(\tilde{u}^n)\|_2 \geq \rho\|F(\tilde{u}^n)\|_2$ *where* $0<\rho<1$*. If such a* k *does not exist, set* $u^n = \tilde{u}^n$ *and go back to step (1).*
   - (b) *Peak-removing. Form a new system of equations; solve it by one of the Jacobi methods using* $\tilde{u}^n$ *as the initial guess; call the solution* v^n*.*
   - (c) *Substitution. If* $\|F(v^n)\|_2 < \|F(\tilde{u}^n)\|_2$*,* $u^n = v^n$*. Or else* $u^n = \tilde{u}^n$*. Go back to step (1).*

One can adjust parameter ρ so that only truly unbalanced nonlinearity in the context of the problem under consideration will be treated by this process. Next, we'll introduce two Jacobi methods to be used with the above algorithm.

2.1 A Zeroth-Order Jacobi Method

The standard nonlinear Jacobi method on one of the subdomains can be described using the global Newton's method on the following system:

$$\begin{cases} H_k^n(u) = F_k(u) \qquad = 0, \\ \\ H_i^n(u) = u_{\Omega_i} - \tilde{u}_{\Omega_i}^n = 0, \ i \neq k, \end{cases} \tag{10}$$

i.e., it approximates the equation $G_i^n(u) = F_i(u) - F_i(\tilde{u}^n) = 0$ for $i \neq k$, with

$$H_i^n(u) = u_{\Omega_i} - \tilde{u}_{\Omega_i}^n = 0.$$

That is, it uses piecewise constant to approximate the solution for $i \neq k$, where k is the index of a subdomain where u_{Ω_k} is considered 'bad', e.g., having the largest residual $\|F_k\|_2$.

An advantage of this approach is, except on Ω_k, that all equations are trivial; i.e., no calculation is necessary to obtain the solution.

2.2 A First-Order Jacobi Method

To remedy the crude approximation used in the zeroth-order Jacobi method, we form a new system of equations,

$$\begin{cases} H_k^n(u) = F_k(u) & = 0, \\ H_i^n(u) = \dfrac{\partial F_i(\tilde{u}^n)}{\partial u}(u - \tilde{u}^n) = J_i(\tilde{u}^n)(u - \tilde{u}^n) & = 0, \ i \neq k, \end{cases} \tag{11}$$

i.e., it approximates the equation $G_i^n(u) = F_i(u) - F_i(\tilde{u}^n) = 0$ for $i \neq k$, with

$$H_i^n(u) = \frac{\partial F_i(\tilde{u}^n)}{\partial u}(u - \tilde{u}^n) = J_i(\tilde{u}^n)(u - \tilde{u}^n) = 0.$$

In other words, it approximates the difference in F_i at $\tilde{u}^n$ using its differential at $\tilde{u}^n$.

We remark here that the algorithms just proposed are similar in spirit to the recently introduced nonlinear preconditioning algorithms [4,5,9], but the nonlinear treatment here is more local.

3 Numerical Experiments

To demonstrate the efficiency and effectiveness of the new algorithms, we present some numerical results for solving the incompressible Navier-Stokes equations with

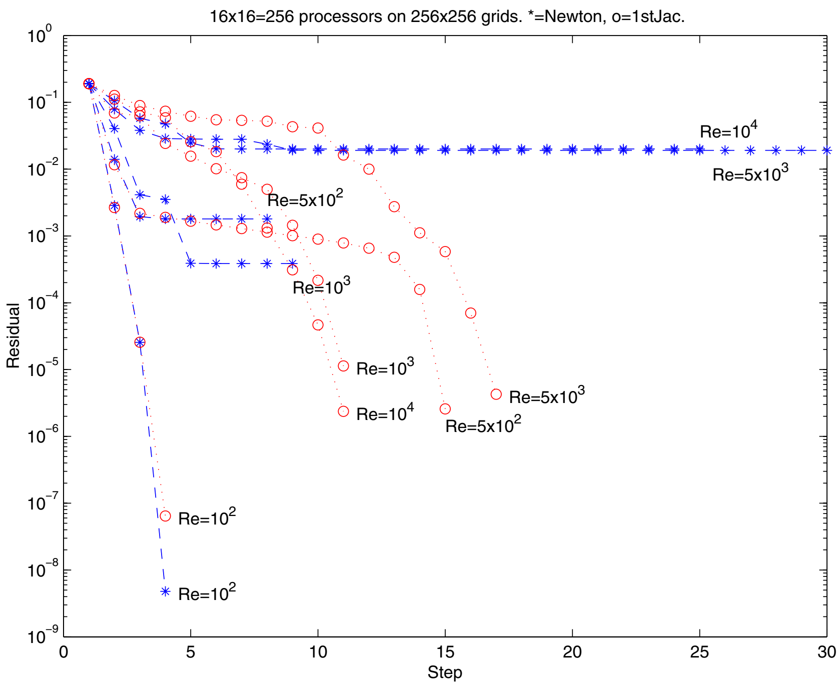


Fig. 1. Nonlinear residual history with varying Reynolds numbers. The classical inexact Newton is marked with $*$ and the new algorithm is marked with $\circ$.

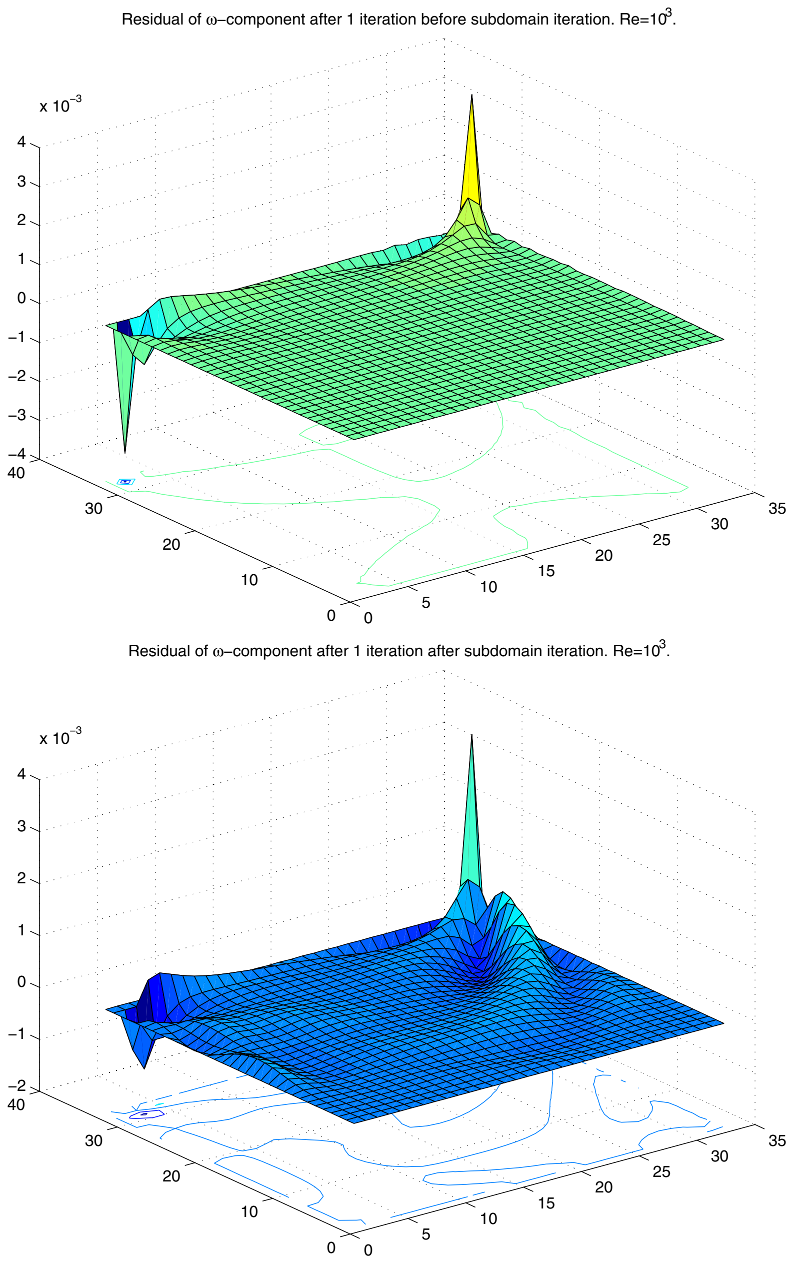


Fig. 2. The effect of subdomain Jacobi iterations: before (top) and after (bottom). One of the corner singularities is chosen to be eliminated by the inner Jacobi iterations.

different Reynolds numbers. The implementation of the proposed algorithms is done using PETSc [1], and the results are obtained on an IBM BG/L. Both the zeroth order and the first order algorithms are implemented, but the first order algorithm is consistently better than the zeroth order algorithm, therefore, we will only report some preliminary experimental results of the first order algorithm.

The results are obtained for solving (1) on a 256×256 mesh with 256 processors. In all tests, the initial iterate is zero for u, v and ω. We stop the global and local nonlinear iteration if

$$\|F(u^n)\| \le \epsilon_{\mathrm{r}} \|F(u^0)\|$$

where $\epsilon_{\mathrm{r}} = 10^{-6}$ and 10^{-5} are used for the global Newton iterations and the SJN iterations, respectively. The linear iteration for solving the global Jacobian system is stopped if the relative tolerance

$$\|F(u^n) - F'(u^n)M_n^{-1}\left(M_n p^n\right)\| \le \eta_{\mathrm{r}} \|F(u^n)\|$$

is satisfied with $\eta_{\mathrm{r}} = 10^{-4}$. The absolute tolerances for all iterations are 10^{-10}. Here M_n^{-1} is an additive Schwarz preconditioner [14] constructed using the Jacobian matrix F', and a partition of computational domain into 256 subdomains in a checker board fashion. The overlapping size is set to 2, and the subdomain linear systems are solved by LU factorization.

In Fig. 1, we show the history of the norm of the residual of several test runs with different Reynolds numbers using the classical inexact Newton's method [3] (marked with "$*$") and the new algorithm (marked with "$\circ$"). As the Reynolds number increases, the nonlinear system becomes more difficult to solve. In fact, the classical method with the standard line search fails to converge once the Reynolds number exceeds 10^2. We did not try to employ other techniques, such as pseudo-time stepping [10] or parameter/mesh continuations [15,16], to improve the convergence of the classical method. On the other hand, SJN converges for a much larger range of Reynolds numbers as shown in Fig. 1 without employing any of the special tricks. In Fig. 2, we show an example of the surface plots of the residual function corresponding to the vorticity of the flow. The top picture is before a local high nonlinearity is removed, and the bottom picture shows the plot after the local Jacobi solver is applied to partially remove the local high nonlinearity.

References

1. Balay, S., Buschelman, K., Gropp, W., Kaushik, D., Knepley, M., McInnes, L., Smith, B., Zhang, H.: PETSc Users Manual, Argonne National Laboratory (2008)
2. Barker, A., Cai, X.-C.: NKS for fully coupled fluid-structure interaction with application. Lecture Notes in Computational Science and Engineering. Springer, Heidelberg (2009)
3. Cai, X.-C., Gropp, W.D., Keyes, D.E., Melvin, R.G., Young, D.P.: Parallel Newton–Krylov–Schwarz algorithms for the transonic full potential equation. SIAM J. Sci. Comput. 19, 246–265 (1998)

4. Cai, X.-C., Keyes, D.E.: Nonlinearly preconditioned inexact Newton algorithm. SIAM J. Sci. Comput. 24, 183–200 (2002)
5. Cai, X.-C., Li, X.: Inexact Newton's methods with nonlinear restricted additive Schwarz preconditioning for problems with high local nonlinearities (in preparation)
6. Dennis Jr., J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM, Philadelphia (1996)
7. Eisenstat, S.C., Walker, H.F.: Globally convergent inexact Newton methods. SIAM J. Optimization 4, 393–422 (1994)
8. Hirsch, C.: Numerical Computation of Internal and External Flows. John Wiley & Sons, New York (1990)
9. Hwang, F.-N., Cai, X.-C.: A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations. J. Comput. Phys. 204, 666–691 (2005)
10. Kelley, C.T., Keyes, D.E.: Convergence analysis of pseudo-transient continuation. SIAM J. Num. Anal. 35, 508–523 (1998)
11. Lanzkron, P.J., Rose, D.J., Wilkes, J.T.: An analysis of approximate nonlinear elimination. SIAM J. Sci. Comput. 17, 538–559 (1996)
12. Ortega, J., Rheinboldt, W.: Iterative Solution of Nonlinear Equations in Several Variables. SIAM, Philadelphia (2000)
13. Ovtchinnikov, S., Dobrian, F., Cai, X.-C., Keyes, D.: Additive Schwarz-based fully coupled implicit methods for resistive Hall magnetohydrodynamic problems. J. Comput. Phys. 225, 1919–1936 (2007)
14. Smith, B.F., Bjørstad, P.E., Gropp, W.D.: Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, Cambridge (1996)
15. Young, D.P., Melvin, R.G., Bieterman, M.B., Johnson, F.T., Samant, S.S.: Global convergence of inexact Newton methods for transonic flow. Int. J. Numer. Meths. Fluids 11, 1075–1095 (1990)
16. Young, D.P., Mervin, R.G., Bieterman, M.B., Johnson, F.T., Samant, S.S., Bussoletti, J.E.: A locally refined rectangular grid finite element method: Application to computational fluid dynamics and computational physics. J. Comput. Phys. 92, 1–66 (1991)

Workshop on Bioinformatics' Challenges to Computer Science

Bioinformatics' Challenges to Computer Science: Bioinformatics Tools and Biomedical Modeling

Mario Cannataro[1], Rodrigo Weber dos Santos[2], and Joakim Sundnes[3]

[1] University Magna Graecia, Catanzaro, Italy
cannataro@unicz.it
[2] Federal University of Juiz de Fora, Brazil
rodrigo.weber@ufjf.edu.br
[3] Simula Research Laboratory, Norway
sundnes@simula.no

Abstract. The second edition of the workshop on Bioinformatics' Challenges to Computer Science aimed to discuss the gap between bioinformatics tools and biomedical simulation and modeling. This short paper summarizes the papers accepted for the workshop, focusing on bioinformatics applications at the genomics and molecular level as well as simulation and management at the biomedical level, and gives a brief outlook on future developments.

Keywords: Bioinformatics, Data Management and Integration, Modelling and Simulation of Biological Systems, Data Visualization.

1 Bioinformatics - An Overview

Bioinformatics[1,2] is an interdisciplinary field linking biology with computer science. In the strictest interpretation of the term, see e.g. *http://www.wikipedia.org*, bioinformatics is defined as the application of computer science to advance the field of molecular biology. However, for this workshop we have adopted a broader definition, which covers development of advanced algorithms and computational tools to study biomedical problems in general. In addition to classical bioinformatics, this includes the development and use of mathematical models to describe the aggregate behavior of cells, tissues, organs and organ systems, a field often referred to as *systems biology* or *integrative biology.*

Bioinformatics is a field under rapid development, as improvements of both computer hardware, algorithms and data acquisition allow the construction and analysis of increasingly complex and detailed models and datasets. However, although substantial progress has been made, the potential benefit from computer science tools in biomedical science is huge, and still far from being fully utilized. Continuing along the track started with its first edition [3], the current edition of the workshop on Bioinformatics' Challenges to Computer Science aims to discuss the gap between bioinformatics tools and biomedical simulation and modeling.

2 Goals

The aim of this workshop was to bring together scientists from computer and life sciences, to discuss future directions of research. As noted above, our broad definition of the bioinformatics term essentially divides current research into two categories; one dealing with data management and analysis on the genetic and molecular scale, and another dealing with development and application of advanced system level models. While both of these branches have obtained considerable progress in the past, and may continue to do so for many years to come, a closer interaction between the two groups hold an even larger potential.

3 Workshop Summary

The papers of the present workshop cover a broad range of applications. The first session is devoted to applications of simulation studies and signal analysis. Indolfi *et al.* focus on the role of real-time image processing in medicine and present a novel software tool able to assist the physician during pre-implant analysis and thus supporting an optimal stent choice. The paper by Peters *et al.* deals with the application of electrical impedance tomography to assess cardiac ejection fraction. Numerical methods based on biophysical models are used to analyze the potential of this tool. The third paper of the session, by Palumbo *et al.*, describes the integrated analysis of different biomedical signals of the lower limb to characterize various physical exercises. Muscles activity and training effectiveness were evaluated by monitoring electromiography (EMG) signals, metabolic data, oxygen uptake and heart rate, with the overall aim to develop a system able to manage different information coming by various electronic devices.

The papers of the second session focus on the application of computer science tools in genetics and molecular biology. The paper by Zhang *et al.* focuses on an important research problem in DNA microarray data analysis, the discovery of gene co-regulatory relationships, and describes an improved algorithm for the analysis of gene co-regulation. Menif *et al.* address the Haplotype Assembly Problem (HAP), an important aspect of Single Nucleotide Polymorphisms (SNP) analysis, and present a set of algorithms for Minimum Fragment Removal based on the processing of strings. The paper by Miceli *et al.* addresses the protein structure prediction problem, central for the understanding of protein function, and compares several available automatic secondary structure prediction tools.

In addition to the listed papers, the workshop includes time slots for discussion of future directions in bioinformatics. These slots will be used for discussing potential future development of the presented research results, with particular focus on potential directions of mutual benefit and joint interest.

4 Conclusions and Outlook

Bioinformatics tools hold a huge potential for use in medical research and clinical practice. Both the analysis of genetic information offered by classical bioinformatics and the study of systems behavior with detailed mathematical models may lead to huge benefits for drug development and personalized health care. However, the full potential of the field can only be achieved by successfully combining these two branches of the field. Aquisition and analysis tools for genetic data may be combined with system level models to perform patient specific computer simulations of organs and organ systems. This link from the *Genome* to the *Physiome* (see for instance the NSR (National Simulation Resource) Physiome Project - http://nsr.bioeng.washington.edu/) has been defined as a future grand challenge of biomedical research, and is subject of substantial research world wide. In the discussion slots of the current workshop we will attempt to provide a link between the presented research results, and to identify potential collaborative projects that would help to bridge the gap between the two branches.

Acknowledgements. We would like to thank the members of the program committee for the invaluable assistance on reviewing the papers and the organizers of ICCS for promoting this workshop.

References

1. Jones, N.C., Pevzner, P.A. (eds.): An Introduction to Bioinformatics Algorithms. MIT Press, Cambridge (2004)
2. Cohen, J.: Bioinformatics - an introduction for computer scientists. ACM Computing Surveys 36, 122–158 (2004)
3. Cannataro, M., Romberg, M., Sundnes, J., Santos, R.W.d.: Bioinformatics' challenges to computer science. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 67–69. Springer, Heidelberg (2008)

Cartesio: A Software Tool for Pre-implant Stent Analyses

Ciro Indolfi, Mario Cannataro, Pierangelo Veltri, and Giuseppe Tradigo

University Magna Græcia of Catanzaro, Germaneto 88100, Italy
surname@unicz.it
http://www.unicz.it

Abstract. Quantitative angiography has becoming largely used in interventional cardiology thanks to positive outcomes in terms of patients survival indexes and life quality. This technique is based on positioning a medical stent in a laparoscopic intervention to cure coronary stenoses. Patients gain an immediate benefit from such a procedure by avoiding an open-heart surgery procedure. Stent positioning is guided by using contrast liquid and angiographic X-Rays images, which are used to define stent dimensions. Physicians may have difficulties in optimally estimating the stenosis size in order to choose the most appropriate stent mainly because there is no absolute reference on the angiographer screens. In this paper we present an innovative software tool able to assist the physician pre-implant analysis and thus supporting an optimal stent choice.

1 Introduction

To date, angiography has been the primary tool to asses procedural outcomes after a number of cardiac surgery such as: coronary angioplastic or stent implantation. Its principal role has been to identify major complications (dissections, thrombus) resulting from the procedure. The agioplastic cardiology has changed a lot in the last 40 years. It now makes use of highly advanced clinical and surgery techniques (i.e. laparoscopy, angiographic imaging, biomedical stents). One of the most critical factors of this type of surgery is that patient is awake and conscious of what is going on around him during each step of the clinical activity, during both the disease discovery and the surgery phase. This implies that the patient will immediately feel improvements or aggravations of his health condition and this suggests that decisions should be taken faster than what happened in the past. Nevertheless, in order to avoid stent thrombosis, drug eluting stents, as well as bare metal ones, should be perfectly deployed. Thus the estimation of diameter and length of the coronary vessels as well as overlap of stents at the origin of large collateral branches, is critical. Currently, optimization techniques to physicians in order to measure coronary stenoses are included in software tools coming with angiography equipment. Nevertheless such software tools are either too expensive or difficult in use since the estimation is done via a dedicated catheter to be used just for this application.

In this paper we present *Cartesio*, an innovative software tool to be used by physicians working in emodynamics surgery rooms. It helps in making a pre-implant analysis for the estimation of the dimensions of the stent to be implanted. The tool interacts with virtually any angiographic equipment by acquiring its high-resolution video signal and offering a set of functions to zoom, pan, playback, measure and draw a virtual stent over the acquired video frames. Each rendered measurement or stent pre-implant analysis can be exported as a bitmap image on the file system or saved in an experiment repository on a relational database for future reference. The software uses a new patented balloon catheter with radio-opaque iridium markers positioned at 10 mm from each other.

Cartesio allows the operator to calibrate the images by making the operator click over two radio-opaque markers. After the angiographic images are calibrated the operator can use a software measurement tool to measure distances in millimeters instead of pixels. Furthermore, once the calibration is performed, all of the parameters of the virtual stent (i.e. diameter, length) can be expressed in millimeters. Measurements help physicians to evaluate the exact dimension of the stenosis. After determining such a dimension the physician can set the parameters of the virtual stent and visualize it over the vessel structure. The virtual stent is rendered in real time and can be moved along a previously drawn curve path following the desired vessel. The physician have a precise preview of what will be implanted and, if desired, he can change some parameters or move the stent to better fit the disease. By using such a procedure he can take better and more conscious decisions before the real angioplastic surgery.

A clinical advantage of the system is the visual analysis of radio-opaque markers to establish the stent length and the virtual reconstruction of coronary stent in the angiographic image allowing a further visual double-check analysis before the device is irreversibly implanted.

In this paper we describe the software main features, its architecture and its use in an emodynamics surgery room. Physicians can control the *Cartesio* tool via a wireless trackball mouse installed at the patient table and visualize its output in one of the monitors connected to the angiographer.

2 Related Works

During the last ten years many researchers have exploited novel techniques to extract informations from image data acquired via angiographic equipment. These techniques allow to build models used by physicians in order to take better decisions or to execute calculations (i.e. distance evaluations, angles, vessels diameter) and simulations (i.e. blood flow changes, blood pressure, impact of a vessel dissection).

One of these techniques is Virtual Reality which is used to build geometrical models of real objects in order to perform complex simulations. Once a virtual object has been placed in a virtual scene, often called virtual world, a scientist can define physical parameters both for the object (i.e. mass, color, material) and for the world (i.e. gravity, humidity, electric field). Performing simulations in the virtual world is usefull because results are easily reproducible and no real object is involved. Furthermore simulation results can be used to confirm or reject hypotheses. In [14] a complete study about the impact of Virtual Reality simulations over clinical practice is described. Fourty-five experienced interventionalists were recruited and asked to use a commercially available VR simulator to perform a carotid artery stent (CAS) procedure. All subjects rated the simulator highly in terms of realism and training potential and the investigator was able to correlate total and fluoroscopic time of the procedures to the physician's experience level.

Similar investigations about the value of a computer-aided clinical procedures has been explored in various areas of the clinical activity. As an example, we report how similar approaches have been exploited in the neurosurgery field, where biomedical stents are used to cure aneurysms. In [10] a technique for pretreatment planning and visualization of a virtual stent across the aneurysm neck is presented. The authors state that suh a method provides information not otherwise available regarding the location of portions of the stent not visible on fluoroscopy. Furthermore, using the method during a treatment, the operator shows an enhanced ability to determine the location of coils in relation to the stent boundaries, thus avoiding parent artery compromise. In [13] the authors simulate the fluid dynamics effects of positioning multiple stents in the case of an aneurysm in order to prevent a rupture risk. The geometry of a wide-necked saccular basilar trunk aneurysm was reconstructed from a patient's computed tomographic angiography images and three models of commercially available stent were used during simulations. The study showed that The complex flow pattern observed in the unstented aneurysm was suppressed by stenting and the effect was increased by deploying multiple stents.

In literature we can find many papers describing techniques to enhance stent visualization, both in place or virtually, because this has proven to be useful to the physician in deciding how to treat a disease or to evaluate the state or the evolution of a treatment. In [8] the visualization of different coronary artery stent is compared with respect to the detectability of in-stent stenoses during computed tomography in a plastic vessel phantom. The study shows how 16-slice computed tomography is necessary for this task if compared to 4-slice performances. In [9] the ability to assess the coronary artery lumen is measured in the presence of coronary artery stents in multislice spiral computed tomography. In [11] the authors compare some postprocessing techniques for three-dimensional computed tomography in order to visualize normal arterial branches, measure aneurysm diameters and neck lengths as well assessment of vessel patency and presence of endoleaks. In [7] the accuracy of 16-row multidetector computed tomography is measured when used to visualize peripheral artery stents and to the appraise in-stent stenoses. In [12] a computer program is exploited to plan stent-grafting for thoracic aortic aneurysm with complicated morphology. The program, called Semi Automatic Virtual Stent (SAVS) designer, uses three-dimensional computed tomography data to allow the design of a virtual stent which can be used as a guide to shape a real straight stent to be implanted.

The IVUS technique [6] is becoming largely used in the emodynamics surgery and allows an estimation of the vessel dimension and shape by using a dedicated catheter that has to be inserted before angioplastic stent positioning. This technique, even if is currently considered the gold optimization technique, has two main disadvantages: it is expensive and requires a pre-analysis phase that has to be performed before inserting the guide with the guide catheter. The use of *Cartesio* and the radio-opaque marked catheter has the advantage of performing both measurements and guide inserting in the same phase.

Most of the times the angiographer can generate planar images, which means that the physician has to choose the most reliable planar projection of the vessel in order to perform reliable measurements. In order to solve this problem, many efforts have been made by the research community to extract three dimensional models from a pair of planar images taken via the angiographer at different angles. This has lead to two main results: (*i*) the extraction of a tree of vessels representing connections among vessels and their position in space [3,2]; (*ii*) the creation of a full 3D model with polygons describing vessels narrowings and enlargements, useful to execute punctual simulations of blood flow or just to better visualize stenoses from inside the vessel [1,4].

Currently *Cartesio* has been implemented on planar images and the problem of coronary vessels that are not exactly projected on the image plane can be measured by using local calibration that is always guided by radio-opaque catheter that follows the vessel.

Finally many software tools have been defining by major vendors. For instance the edge-detection support tool from the General Electric workstation allows to measure the diameter and the variation of stenosis profile in the vessels.

3 The *Cartesio* Tool

3.1 Requirements and Functionalities

The main phases followed by a phisician during the cardiac surgery activity are:

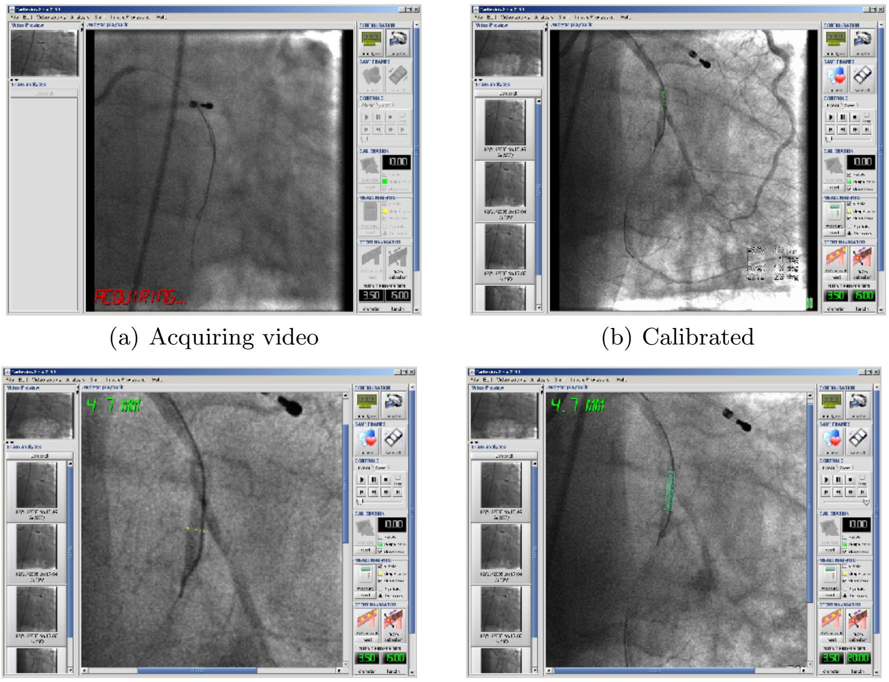
patient disease discovery phase during which he inserts a catheter through the femoral artery to the heart and, by the injection of a contrast medium to distinguish the vessels shapes, he acquires a set of video sequences with different projections;

disease evaluation phase where the phisician analyzes all the video sequences and frames in order to elaborate and possibly verify a clinical hypothesis on the disease;

plan of action phase in which the medical equipe decides what to do and how, evaluating potential problems and consequences and minimizing the decision time;

intervention phase where the surgery activity is eventually implemented.

The tool has been designed following such phases and offers useful functionalities over the flat angiographic image acquisitions, such as: video signal acquisition, video live, storing, zooming with bi-cubic interpolation, calibration,



(a) Acquiring video (b) Calibrated

(c) Measuring (d) Stent drawing

Fig. 1. Four interaction phases with the *Cartesio* tool

measurements of the coronary diameter and length. In addition the software is also able to reconstruct in the coronary angiographic segment the stent that operator have selected for a specific lesion. Moreover, it is required to store the data acquired during analysis in the Dicom standard image format on a local repository, data that can be reused for case studies. For instance, in figure 1, screenshots related to the Acquisition, Calibration, Measuring and Stent drawing and positioning simulation phases are depicted.

The software tool has also been developed to perform procedures both on-line and off-line. In the first case the software tool allows to calibrate and measure vessels and coronaries interactively during the procedure. The second case allows to perform measurements on the images stored in a standard DICOM[1] repository. Many Interventional Units have a shared Picture Archiving and Communication System (*PACS*) able to store and retrieve medical images from a large number of equipment in the DICOM format. With the ability to use images stored in the DICOM format, *Cartesio* offers physicians the ability to perform simulations even after the coronarography analysis, simulating stent positioning and deciding strategies for future treatments.

[1] Digital Imaging and Communications in Medicine (*DICOM*) is a standard for handling, storing, printing, and transmitting information in medical imaging. It includes a file format definition and a network communications protocol.

Moreover the *Cartesio* tool includes classical image manipulation techniques such as: acquiring a video sequence from the angiographer, choosing the best contrast frame on which build the virtual stent model, calibrating the image, measuring distances (i.e. vessel diameters, stenosis diameters), designing the vessel path, choosing the stent parameters and interactively moving it along the designed path.

3.2 Architecture

The system architecture is structured as a set of components all connected together through a communication bus, as depicted in figure 2.

Each component of the system implements a common interface enabling it to send and receive messages from the bus. In the initialization phase of the module, it searches for the bus object and makes a subscription. Every other module connected to the bus receives notifications in a publisher/subscriber fashion and our API provides both the pull and the push models. The main *Cartesio* modules are:

Communication Bus is used by all the other modules to exchange messages and notify events in a publisher/subscriber fashion; when a module initializes itself it registers with the Communication Bus in order to receive and send messages to the other modules;

Command Interpreter together with the User Interface module it is in charge of transforming commands issued by the operator into instructions to be send the other *Cartesio* modules;

Rendering and Business Logic module implements the logic of the system and offers math, geometric and measuring functionalities to other modules (i.e. conversion functions to transform pixel into millimeters and back); furthermore it holds the state of the system, the data structures and periodically generates rendered images to be shown on the screen by combining bitmap angiographic images with vector data (i.e. curve paths, lines, points, measurement labels);

Video Acquisition module works with both JMF (*Java Media Framework*) video sources (i.e. low resolution USB video signal) and distributed digital video streams (i.e. native code video components sending acquired video over a TCP/IP network);

Persistence module offers functionalities to import exams in the DICOM format, both for didactica and post-therapy planning, and to export data both on file system, in the XML format, and on a relational database (*R-DBMS*);

Cartesio is part of a *Distributed Electronic Patient Record*, in which the patient's personal data and family history are enriched with clinical files and DICOM images documenting exams results. The patient files are included during the coronarography planning and all files with biological data and DICOM images are included in the Electronic Patient Record for future reference.

Main modules are depicted in figure 2. In the following we briefly discuss the Video Acquisition module and we sketch the Rendering module as the main

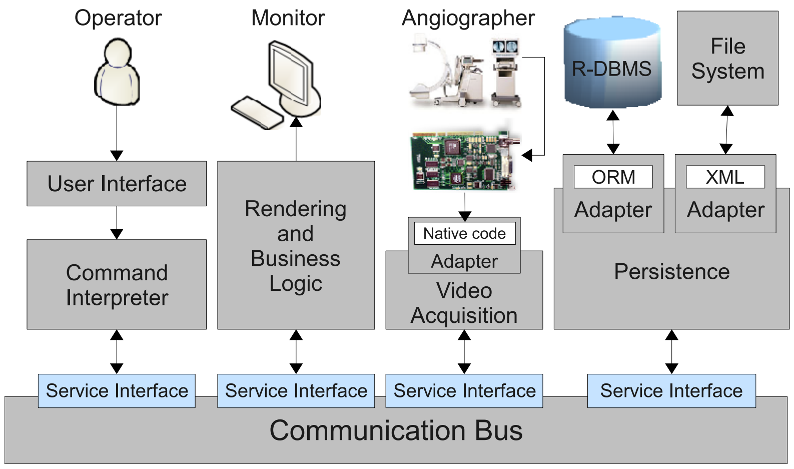


Fig. 2. Module architecture of the *Cartesio* tool

algorithmically challenging. The acquisition module consists in acquiring an high resolution and high frequency analog video signal generated from angiographic equipment. The lack for a video signal standard in medical applications have lead to a plethora of different specifications, which makes really hard to design an acquisition platform working for every equipment, even for instruments from the same vendor. We managed to acquire all of the video signals generated by equipment found in our structure, by using the following strategy:

1. measuring peak-peak voltage, identifying the video sync signal and resolution;
2. identifying the acquisition signal frequency;
3. tuning the acquisition module in order to correctly hook the video signal and in order to acquiring data in main memory.

The data volume generated during the video signal digitalization is, in generale, very large. For instance, for the General Electrics Innova System, the acquired full-resolution video signal generates a data flow of approximately 20 Megabytes per second. For this reason, while Cartesio has been implemented using the Java language, the video acquisition component has been coded in native code in order to maximize the frame rate. After the signal has been correctly grabbed and digitalized, it is made available to *Cartesio*. Note that, due to the large amount of data, an ad-hoc communication channel between the image acquisition native component and the *Cartesio* image manipulation module has been designed.

The Rendering module is in charge to position and visualize vector data over the bitmap angiographic images in *Cartesio*. It interpolates the stent path, designed by the physician across a vessel of interest, through a set of Bezier curves [5] connecting a set of control points. By moving or adding control points on the angiographic image, the interpolation curve is recalculated in real time. Along the curve path the operator will be able to move and resize the virtual stent. The stent shape will be visible after the definition of the stent diameter and length, by calculating two bezier curves equidistant from the main stent path.

4 Experiments and Conclusions

Since June 2008 the *Cartesio* tool has been testing in Emodynamic Department of University "Magna Græcia" of Catanzaro Medical School Hospital. As a first setup *Cartesio* has been installed on a 2Gb RAM quad core computer with Microsoft Window operating system. The video signal has been acquired from the video output of an Rx machine General Electrics Innova 2100. The system has been installed in the emodynamics surgery room and can be controlled both by the physician during the interventional procedure or by a technician outside the operating room. According to our tests, *Cartesio* measurement facilities and simulations improve the evaluation of correct stent choice thus optimizing the interventional procedure in terms of time and confidence. The improved capability in correctly evaluating the stent dimensions has been confirmed by expert operators. The only drawback is mainly that the physician has to choose a correct projection for acquiring the angiographic images (i.e. planar w.r.t. the diseased coronary) in order to allow uniform measurement analyses through the whole image. The software has also been tested with the italian SIAS angiographer and also on the General Electrics Advantic, with similar performances.

We conclude claiming that *Cartesio* is a software tool that both enhances the real-time visualization of the angiographic images and offers new image processing functionalities supporting the surgery procedures. Currently the tool is in use in two surgery rooms in the University of Catanzaro Medical School Hospital, and data have been storing in the *Cartesio* DICOM database. We are improving the software tool by adding multiple point calibration and designing a collaborative version allowing the operator to share the virtual stent simulation environment with remote collegues. Such a collaborative environment could bring new emodynamic techniques to remote hospital or clinical structures not having the expertise in using novel or advanced angioplastic techniques.

References

1. Yang, J., Wang, Y.T., Liu, Y., Tang, S.Y., Li, Y.H.: 3-D Reconstruction of Coronary Arterial Trees from Uncalibrated Monoplane Angiographic Images. Chinese Journal of Computers 30, 1594–1602 (2007)
2. Kerrien, E.: Outils d'imagerie multimodalité pour la neuroradiologie interventionnelle. Phd Thesis, Institut National Polytechnique de Lorraine (2007)
3. Kerrien, E., Vaillant, R., Launay, L., Berger, M.O., Maurincomme, E., Picard, L.: Machine precision assessment for 3D/2D digital subtracted angiography images registration. In: Hanson, K. (ed.) Proceedings of SPIE Medical Imaging, vol. 3338, pp. 39–49 (1998)

4. Ogiela, M.R., Tadeusiewicz, R., Trzupek, M.: Graph-Based Semantic Description in Medical Knowledge Representation and 3D Coronary Vessels Recognition. In: Indulska, J., Ma, J., Yang, L.T., Ungerer, T., Cao, J. (eds.) UIC 2007. LNCS, vol. 4611, pp. 1079–1088. Springer, Heidelberg (2007)
5. Knuth, D.: Metafont: the Program, pp. 123–131. Addison-Wesley, Reading (1986)
6. Safian, R.D., Freed, M.S.: Intravascular Ultrasound. The Manual Of Interventional Cardiology, 712. Physicians Press (2002)
7. Herzog, C., Grebe, C., Mahnken, A., Balzer, J.O., Mack, M.G., Zangos, S., Ackermann, H., Schaller, S., Seifert, T., Ohnesorge, B., Vogl, T.J.: Peripheral artery stent visualization and in-stent stenosis analysis in 16-row computed tomography: an in-vitro evaluation. European Radiology 15, 2276–2283 (2005)
8. Maintz, D., Seifarth, H., Flohr, T., Kramer, S., Wichter, T., Heindel, W., Fischbach, R.: Improved Coronary Artery Stent Visualization and In-Stent Stenosis Detection Using 16-Slice Computed-Tomography and Dedicated Image Reconstruction Technique. Investigative Radiology 38, 790–795 (2003)
9. Mahnken, A.H., Buecker, A., Wildberger, J.E., Ruebben, A., Stanzel, S., Vogt, F., Gunther, R.W., Blindt, R.: Coronary Artery Stents in Multislice Computed Tomography: In Vitro Artifact Evaluation. Investigative Radiology 39, 27–33 (2004)
10. Karmonik, C., Strother, C.M., Chen, X., Deinzer, F., Klucznik, R., Mawad, M.E.: Stent-Assisted Coiling of Intracranial Aneurysms Aided by Virtual Parent Artery Reconstruction. American Journal of Neuroradiology 26, 2368–2370 (2005)
11. Sun, Z., Winder, R.J., Kelly, B.E., Ellis, P.K., Kennedy, P.T., Hirst, D.G.: Diagnostic Value of CT Virtual Intravascular Endoscopy in Aortic Stent-Grafting. Journal of Endovascular Therapy 11, 13–25
12. Hyodoh, H., Katagiri, Y., Sakai, T., Hyodoh, K., Akiba, H., Hareyama, M.: Creation of individual ideally shaped stents using multi-slice CT: in vitro results from the semi-automatic virtual stent (SAVS) designer. European Radiology 15, 1623–1628 (2005)
13. Kim, M., Levy, E.I., Meng, H., Hopkins, L.N.: Quantification of Hemodynamic Changes Induced By Virtual Placement of Multiple Stents Across A Wide-Necked Basilar Trunk Aneurysm. Neurosurgery 61, 1305–1313 (2007)
14. Van Herzeele, I., Aggarwal, R., Choong, A., Brightwell, R., Vermassen, F., Cheshire, N.: Virtual reality simulation objectively differentiates level of carotid stent experience in experienced interventionalists. Journal of Vascular Surgery 46, 855–863 (2007)

Determination of Cardiac Ejection Fraction by Electrical Impedance Tomography - Numerical Experiments and Viability Analysis

Franciane C. Peters, Luis Paulo S. Barra, and Rodrigo Weber dos Santos

Universidade Federal de Juiz de Fora, Juiz de Fora, Minas Gerais, Brasil
franciane.peters@engenharia.ufjf.br, luis.barra@ufjf.edu.br, rodrigo.weber@ufjf.edu.br

Abstract. Cardiac ejection fraction is a clinically relevant parameter that is highly correlated to the functional status of the heart. Today the non-invasive methods and technology that measure cardiac ejection fraction, such as MRI, CT and echocardiography do not offer a continuous way of monitoring this important parameter. In this work, we numerically evaluate a new method for the continuous estimation of cardiac ejection fraction based on Electrical Impedance Tomography. The proposed technique assumes the existence of recent Magnetic Resonance (MR) images of the heart to reduce the search space of the inverse problem. Simulations were performed on two-dimensional cardiac MRI images with electric potentials numerically obtained by the solution of the Poisson equation via the Boundary Element Method. Different protocols for current injection were evaluated. Preliminary results are presented and the potentialities and limitations of the proposed technique are discussed.

1 Introduction

Cardiac ejection fraction indicates the measure of the blood fraction that is pumped from each ventricle in each step of the heart cycle. The ejection fraction of both left ventricle (EFLV) and right ventricle (EFRV) can be determined. But the clinical use of EFLV is more common and it is frequently called ejection fraction (EF). By definition, the ejection fraction is calculated in the following way:

$$EF = \frac{PV}{EDV} = \frac{EDV - ESV}{EDV} \tag{1}$$

where PV is the volume of blood pumped, that is given by the difference between the end-diastolic volume (EDV) and the end-systolic volume (ESV). Cardiac ejection fraction is a relevant parameter that is highly correlated to the functional status of the heart. To determine EF, different techniques can be used, like echocardiography, cardiac magnetic resonance and computed tomography. However, because of the costs of these techniques, they can not be used for continuous monitoring. In this work, we numerically evaluate a new method for the continuous estimation of cardiac ejection fraction based on Electrical Impedance Tomography (EIT).

EIT is a technique that reconstruct conductivity distribution images inside a conductor domain based on protocols of current injection and potential measurement on the boundary of the domain. This technique has a large utilization on geophysics, monitoring of industrial activities [1] and on biomedical engineering [2,3,4]. In this last area, the EIT is considered as a viable technique for monitoring long periods, since it is not based on ionizing radiation. The EIT spacial resolution is not as high as the traditional imaging methods. Nevertheless, its portability, low cost and time resolution are the main advantages of the technique.

The aim of the present work is to present a study on the viability of EIT to the continuous monitoring of cardiac ejection fraction. Previous work have shown preliminary results on the same topic [5,6,7]. In this work, we extend previous results and give new contribution in many aspects: 1) We develop a more realistic 2D model that includes the lungs. Because of their low conductivity, these regions behave as a barrier to the electrical currents and poses new challenges to the problem. 2) The results presented in this work were obtained using the Levenberg-Marquardt method [8] for the solution of the inverse problem associated to EIT. Previous work have adopted different methods, such as the Powell's method [9], Genetic Algorithms [10,11] and FAIPA (Feasible Arc Interior Point Algorithm) [12]. 3) Different protocols for current injection are evaluated for the estimation of cardiac ejection fraction.

Preliminary results are presented and the potentialities and limitations of the proposed technique are discussed. The results suggest the proposed technique is a promising diagnostic tool that offers continuous and non-invasive estimation of cardiac ejection fraction.

2 Methods

2.1 2D Models Based on Magnetic Resonance Images

From magnetic resonance (MR) images, the regions of interest, in this case the two ventricles were manually segmented in two different phase: end of the systole and the end of diastole. Each curve of the segmentation was parameterized by a spline, with a minimum number of points. The left ventricle (LV) spline has 7 control points and the right one (RV) 8 points. The external boundary of the thorax and the boundaries of the lungs were also segmented. For simplicity, these curves are assumed constant during the heart cycle. Figure 1 shows a segmentation example.

The goal of our method is to recover the shape of the internal cavi ties of the heart, presently considered in two-dimensions, from electric potential measurements. Therefore, with two coordinates for each spline control point the methods would need to estimate a total of 30 ((7+8)*2) parameters. To reduce the number of parameters to be estimated the following strategy was adopted.

During MRI segmentation we have used the same number of control points for the splines in both systole and diastole phase. This allows us to restrict the search space forcing that each control point i belongs to a line that connects the position

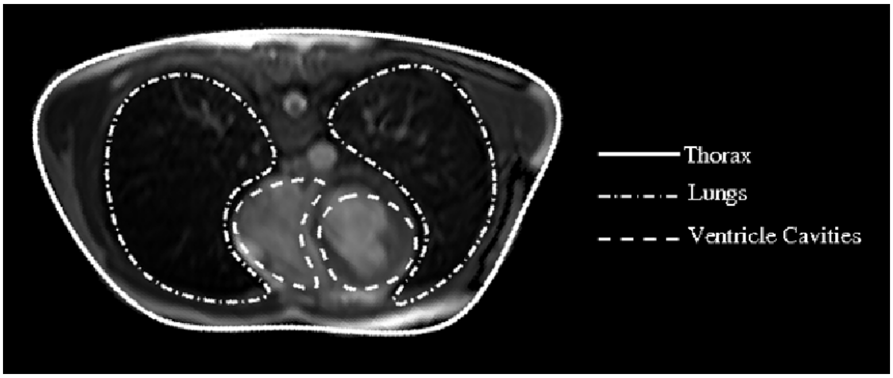


Fig. 1. Manual segmentation of an MR image. The boundary of the lungs is represented in blue and the boundary of the ventricle cavities in red.

at systole and diastole. Thus, a linear interpolation is used, parameterized by a scalar t_i, between the values of the coordinates of each control point i. The spline relative to the end of systole can be obtained with $t_i = 0, \forall i$, and the one relative to the end of diastole with $t_i = 1, \forall i$. Doing so, the method goal is to recover the shape of the ventricular cavities via the estimation of the 15 parameters t_i, with $i = 1...15$.

2.2 Forward Problem and Governing Equations

The proposed 2D model splits the domain in regions that represent different biological tissues, heart cavities, lungs and torso. Each tissue can be mainly electrically identified by its conductivity. Grimnes [13] presents the main factors that influence the properties of biological tissues. Although they may be classified in only four groups, epithelium, muscle, connective tissue and nervous tissue, the tissues can be divided in thirty kinds in accordance to their electrical properties [14]. In addition, the value of the conductivity of each tissue depends on the frequency of the electrical current, on the temperature, on the presence of water, among other issues.

Table 1. Resistivity values of biological tissues that are found in the literature

Tissue	Resistivity (Ωcm)	Reference
	150	Barber and Brown [16]
Blood	150	Yang and Patterson [17]
	100	Schwan and Kay [18]
	400	Patterson and Zhang [19]
Heart	250	Yang and Patterson [17]
	400 - 800	Baysal and Eyuboglu [20]
	727 - 2363	Barber and Brown [16]
Lung	1400	Patterson and Zhang [19]
	600 - 2000	Baysal and Eyuboglu [20]

In this work, we assume the conductivity of each tissue is taken as known, constant and isotropic. These are all simplified assumption, since biological tissues are usually heterogeneous and anisotropic. However, biological tissues are difficult to characterize, and the reported values vary substantially in the literature. Table 1 presents some resistivity values for blood, heart and lung found in the literature.

For the remaining tissues that compose the section of the thorax, that from here on we call the torso region, Bruder et al. [15] proposes a mean resistivity of $500\Omega cm$. The conductivity of the air is $10^{20}\Omega cm$, but the conductivity of the lung filled of air is difficult to determine. Rush et al. [21] presents a very simplified resistivity distribution model characterized by the presence of cavities filled of blood, surrounded by homogeneous material with resistivity ten times greater. The same scheme, properly extended to include the lung regions, is used in this work. Preliminarily, the resistivity of the blood is here taken as $100\Omega cm$ and the torso to be $1000\Omega cm$. Two different values were tested for the resistivity of the lungs: $20000\Omega cm$ (Ratio of Lung to Torso resistivity (RLT) of 20) and $50000\Omega cm$ (RLT of 50).

The forward problem consists of calculating the electrical potential on the external boundary of the torso that is generated by the current injection on a pair of electrodes. Given that our 2D model has three regions with different but constant and isotropic conductivities (heart cavities, lungs and torso) the electrical potential at each point of the regions, ϕ, must satisfy Laplaces' equation:

$$\nabla^2\phi = 0 \tag{2}$$

and the boundary conditions are

$$\begin{aligned}
\sigma_L \nabla\phi &= \sigma_T \nabla\phi \; , & \mathbf{x} &\in \Gamma_1 \\
\sigma_B \nabla\phi &= \sigma_T \nabla\phi \; , & \mathbf{x} &\in \Gamma_2 \\
\sigma_T \frac{\partial\phi}{\partial\mathbf{n}} &= J_i \; , & \mathbf{x} &\in \Gamma_3^{ie} \\
\frac{\partial\phi}{\partial\mathbf{n}} &= 0 \; , & \mathbf{x} &\in (\Gamma_3 - \Gamma_3^{ie})
\end{aligned}$$

where Γ_1 is the interface between the lung and torso region, Γ_2 is the interface between the blood and the torso region, Γ_3 is the external boundary of the thorax, Γ_3^{ie} is the part of Γ_3 where the ith electrode is, J_i is the electrical current injected on the ith electrode and σ_L, σ_B and σ_T are the conductivities of the lung, blood and torso, respectively.

In the present work, the forward problem is solved by the Boundary Element Method (BEM) [22]. Further details of the implementation can be found in [23].

2.3 The Inverse Problem

The inverse problem associated to EIT aims to recover the shape of the ventricular cavities via the estimation of the vector $\mathbf{t}$, that contains the 15 parameters t_i, with $i = 1...15$ (as described in Section 2.1). This is done via the minimization of an objective function that measures the distance between measured electrical potential values ($\bar{\phi}_j$) taken from a pair of electrodes identified by j on the external boundary and the computed ones ($\phi(\mathbf{t})_j$) that depends on the heart cavity shape parameterized by $\mathbf{t}$ and is calculated as described in Section 2.2. Therefore, the goal is to find the best parameter vector $\mathbf{t}$ that minimizes Eq. 3:

$$F = \frac{1}{2}\sum_{j=1}^{m}(\phi(\mathbf{t})_j - \bar{\phi}_j)^2 \tag{3}$$

where m is the number of measurements and depends on the current injection pattern. In this work, the "measured" electrical potential values ($\bar{\phi}_j$) were also synthetically generated, i.e., also numerically obtained.

This minimization problem is solved by the method known as Levenberg-Marquardt [8]. The implementation of the method and the subroutines that compute the objective function are done in Fortran77.

2.4 Numerical Experiments and Stimulation Patterns

For the 2D problem in consideration the areas of the transversal section of the cavities were assumed to be proportional to their volumes, i.e. a cylindrical approximation so that EF is calculate by:

$$EF = \frac{EDA - ESA}{EDA} \tag{4}$$

where EDA and ESA are the areas of the transversal section of the ventricle at the end of the diastole and at the end of the systole, respectively.

From MR images taken at the end of the systole and at the end of the diastole the cardiac ventricles were manually segmented and in accordance to (4) the EF of the left ventricle is 59.24% and the EF of the right ventricle is $29,95\%$. After that, a cardiac disfunction was synthetically generated. The simulated disfunction consists of a modified cardiac cycle in which the end-diastolic volume is the same as in the normal cycle but the end-systolic volume is greater than the normal one. In this new cardiac cycle, the EF of the left ventricle is 33.01% and the EF of the right ventricle is 16.19%. These are the target values to be estimated by the here proposed method.

Two patterns of electrical current injection were tested: a diametrical one, with electrodes evenly distributed along the torso boundary; and an adaptive one, with electrodes that are near the region of interest. The first yields a set of 104 measurements and the second one yields a set of 78. The arrows of the Fig. 2 present the pairs of electrodes sequentially used for current injection in each pattern.

As mentioned before in Section 2.1 we have also tested two different 2D models. Each with a different value for the resistivity of the lungs: $20000\Omega cm$ (Ratio of Lung to Torso resistivity (RLT) of 20) and $50000\Omega cm$ (RLT of 50). Finally, for each of 4 optimization problems (2 Stimulus Patterns times 2 RLT models) we have tested the optimization method with two different initial guesses. One

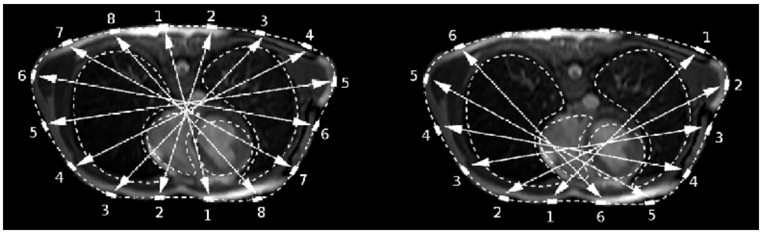


Fig. 2. Two stimulation patterns used in this work. The first one is the diametrical and the second one is the adaptive.

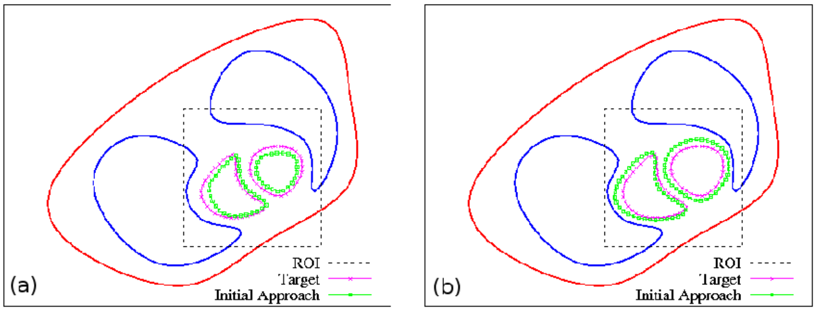


Fig. 3. A typical target (pink) and the two initial guesses (green) given to the optimization procedure: (a) $t_i = 0, \forall i$; (b) $t_i = 1, \forall i$

guess is the parameter set **t** that corresponds to the shape of the ventricles at the end of the diastole of normal heart, i.e. $t_i = 1, \forall i$ and the other at the end of the systole for the normal tissue, i.e. $t_i = 0, \forall i$. The initial guesses and the targets can be compared in Fig. 3. Thus, the method was executed a total of 8 times (2 Stimulus Patterns times 2 RLT models times 2 initial guesses).

3 Results

Table 2 presents the results of our numerical experiments that aims the EF estimation of the synthetically generated cardiac disfunction. The columns present the results for the models with different values for the resistivity of the lungs: Ratio of Lung to Torso resistivity (RLT) of 20 and RLT of 50. Each couple of rows presents the comparison of the two stimulation pattern implemented: diametrical and adaptive pattern. In addition, for each pair (stimulus pattern, RLT) results are presented for two different initial conditions. The first one corresponds to the shape of the ventricles at the end of the diastole of the normal heart, i.e. $t_i = 1, \forall i$ and the other at the end of the systole for the normal heart, i.e. $t_i = 0, \forall i$. The last row of the table presents the target EF values for comparison.

Table 2. Values of the ejection fraction estimated for the synthetic cardiac disfunction for two RLT ratios and two different stimulation patterns. The last row shows the target values.

	RLT= 50		RLT= 20	
Initial	EF (%)		EF (%)	
Guess	RV	LV	RV	LV
Diametrical Pattern				
$t_i = 0$	13.00	34.41	15.32	34.22
$t_i = 1$	16.09	32.21	15.80	33.04
Adaptive Pattern				
$t_i = 0$	12.97	35.86	20.54	29.94
$t_i = 1$	18.72	32.84	20.89	29.40
Target	16.19	33.01	16.19	33.01

Figure 4 allows a geometrical comparison between the final results and the actual target curves. It is important to emphasize that, to make the comparison fair, the results presented in this figures are obtained with the same initial guesses, $t_i = 1, \forall i$.

The results show that in general the error in the computed ejection fraction of the left ventricle is smaller than the one of the right. The mean absolute error of the left ventricle results is 1.64 while the right ventricle ones is 2.42. Moreover, except in one case, the diametrical pattern provides results closer to the actual values than the other pattern. The diametrical pattern provides a mean absolute error of 1.00 while the error of the adaptive pattern is 3.06. About the initial guess, both of them provided good results. But the best ones are obtained with the guess on the original diastole curve with a mean absolute error of 1.54 against an error of 2.52 for the other initial guess.

The geometrical results presented in Fig. 4, showing only the ventricular cavities, suggest that the results become worst in the case the lung resistivity is greater. This behavior is expected because greater resistivities around the region of interest tend to block the electrical current to reach this area. For instance, for the best experimented pattern, the diametrical one, the mean absolute error obtained with the greatest resistivity (RLT = 50) is 2.2 times the error obtained with the other lung resistivity (RLT = 20).

Using the diametrical pattern, the best result was obtained for the left ventricle and RLT of 20. The absolute error in the value of the ejection fraction is of 0.03. It is possible to see this result in Fig. 4(b). In this case it is very difficult to see the difference between the result and the target and it is the best of the 8 tested cases. About the adaptive pattern, the best one was obtained for the left ventricle and RLT of 50. In this case, the absolute error in the value of the ejection fraction is 0.17.

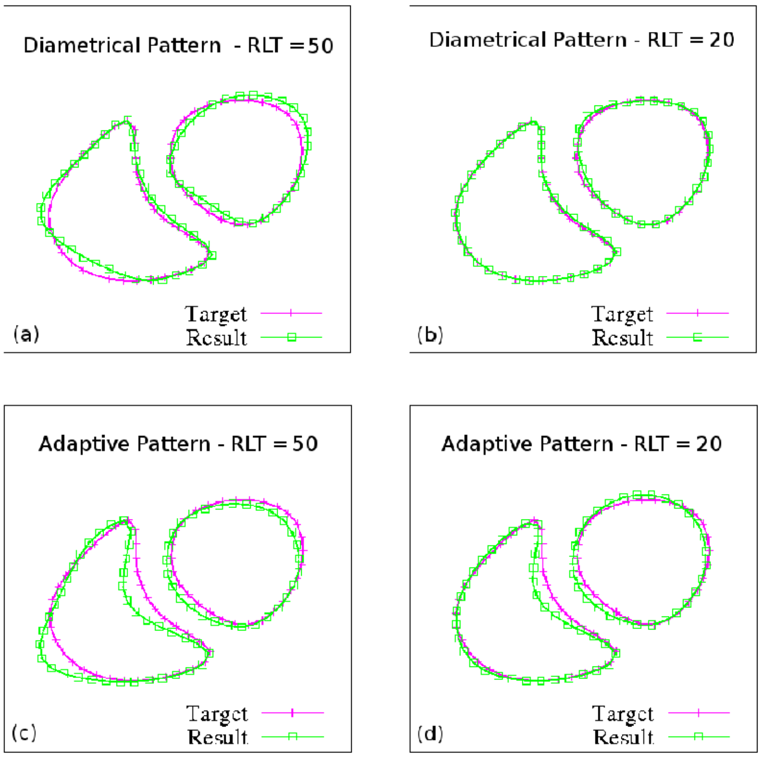


Fig. 4. Some results for the diametrical and the adaptive pattern and the target

4 Conclusions

The presented results suggest that the proposed methodology allows a suitable indication of the cardiac ejection fraction. We observed that the error in the ejection fraction predictions for the right ventricle are greater than those found for the left ventricle and this is in agreement with other techniques, such as with echocardiography.

Concerning the different patterns for current injection tested in this work, the errors obtained with the diametrical pattern are smaller than those using the adaptive pattern, in general. However this fact does not discard the use of the adaptive pattern, as it presents good results and spends around 19 min. in a Pentium 4, 3.00 GHz, for a complete solution, 25% less then the diametrical.

Comparing the results obtained with different lung resistivities we may conclude that the inverse problem becomes more difficult to be solved as the RLT increases. Therefore, the results suggest the current injection should be triggered during the expiratory phase, when the air volume and the corresponding lung resistivity are smaller.

The preliminary results presented in this work suggest the proposed technique is a promising diagnostic tool that may offer continuous and non-invasive estimation of cardiac ejection fraction.

Acknowledgements. The first author would like to thank CAPES and Mestrado em Modelagem Computacional (UFJF) for the Master's scholarship.

References

1. Kim, M., Kim, K., Kim, S.: Phase boundary estimation in two-phase flows with electrical impedance tomography. Int. Comm. Heat Transfer 31(8), 1105–1114 (2004)
2. Polydorides, N., Lionheart, W., McCann, H.: Krylov subspace iterative techniques: On the brain activity with electrical impedance tomography. IEEE Transactions on Medical Imaging 21(6), 596–603 (2002)
3. Seo, J., Kwon, O., Ammari, H., Woo, E.: A mathematical model for breast cancer lesion estimation: Electrical impedance technique using TS2000 commercial system. IEEE Transactions on Biomedical Engineering 51(11), 1898–1906 (2004)
4. Trigo, F., Lima, R., Amato, M.: Electrical impedance tomography using extended Kalman filter. IEEE Transactions on Biomedical Engineering 51(1), 72–81 (2004)
5. Barra, L.P.S., Peters, F.C., Martins, C.P., Barbosa, H.J.C.: Computational experiments in electrical impedance tomography. In: XXVII Iberian Latin American Congress on Computational Methods in Engineering, Belém, Brasil (2006)
6. Barra, L.P.S., Santos, R.W., Peters, F.C., Santos, E.P., Barbosa, H.: Parallel computational experiments in electrical impedance tomography. In: 18th Symposium on Computer Architecture and High Performance Computing, Ouro Preto, Brasil, vol. 1, pp. 7–13. Sociedade Brasileira de Computação, High Perfomance Computing in the Life Sciences (2006)
7. Barra, L.P.S., Mappa, P., Cardoso, S., Peters, F.C.: Comparison of the Computational Performance of Optimization Algorithms in the Solution of an Inverse Problem (in Portuguese). In: VIII Simpósio Mecânica Computacional - SIMMEC, Belo Horizonte, Brasil. PUC-Minas (2008)
8. Madsen, K., Nielsen, H., Tingleff, O.: Methods for non-linear least squares problems (2004)
9. Press, W., Teukolsky, S., Vetterling, W.: Flannery: Numerical Recipes in Fortran, vol. 77. Cambridge University Press, Cambridge (1986)
10. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1978)
11. Holland, J.: How Adaptation builds Complexity. Addison-Wesley Pub. Co., Reading (1995)
12. Herskovits, J.: Feasible direction interior-point technique for nonlinear optimization. Journal of Optimization Theory and Applications 99(1), 121–146 (1998)
13. Grimnes, S.: Bioimpedance and bioelectricity basics (2008)
14. Gabriel, C.: The dielectric properties of biological tissue: I. literature survey (1996)
15. Bruder, H., Scholz, B., Abrahamfuchs, K.: The influence of inhomogeneous volume conductor models on the ECG and the MCG. Physics in Medicine and Biology 39(11), 1949–1968 (1994)
16. Barber, D.C., Brown, B.H.: Applied potential tomography. Journal of Physics E-Scientific Instruments 11(9), 723–733 (1984)

17. Yang, F., Patterson, R.P.: The contribution of the lungs to thoracic impedance measurements: a simulation study based on a high resolution finite difference model. Physiological Measurement 28(7), S153–S161 (2007)
18. Schwan, H.P., Kay, C.F.: Specific resistance of body tissues. Circulation Research 4(6), 664–670 (1956)
19. Patterson, R.P., Zhang, J.: Evaluation of an EIT reconstruction algorithm using finite difference human thorax models as phantoms. Physiological Measurement 24(2), 467–475 (2003)
20. Baysal, U., Eyuboglu, B.M.: Tissue resistivity estimation in the presence of positional and geometrical uncertainties. Physics in Medicine and Biology 45(8), 2373–2388 (2000)
21. Rush, S., McFee, R., Abildskov, J.A.: Resistivity of body tissues at low frequencies. Circulation Research 12(1), 40–50 (1963)
22. Brebbia, C., Telles, J.C.F., Wrobel, L.C.: Boundary Elements Techniques: Theory and Applications in Engineering. Springer, Heidelberg (1984)
23. Barra, L.P.S., Peters, F.C., Santos, R.W.: Numerical Experiments for the Viability Analysis of the Determination of the Cardiac Ejection Fraction by the Electrical Impedance Tomography (in Portuguese). In: XXIX CILAMCE - Congresso Ibero Latino Americano de Métodos Computacionais em Engenharia, Maceió, Brasil. ABMEC (2008)
24. Metherall, P.: Three Dimensional Electrical Impedance Tomography of the Human Thorax. PhD thesis, University of Sheffield (1998)

Analysis of Muscle and Metabolic Activity during Multiplanar-Cardiofitness Training

Arrigo Palumbo, Teresa Iona, Vera Gramigna, Antonio Ammendolia,
Maurizio Iocco, and Gionata Fragomeni

Magna Graecia University, Campus S. Venuta,
88100 Catanzaro, Italy
{palumbo,iona,gramigna,ammendolia,miocco,
fragomeni}@unicz.it

Abstract. Electronic devices have been useful to evaluate muscle fatigue and activation, co-ordination and metabolic consuption among different sports activities. Since these important variables have not been investigated during fitness training, the present study aims to analyze the activity of the major muscles of the lower extremity during training activities on a cardiofitness apparatus (Cardio Wave^{TM}, Technogym$^{®}$ - Gambettole, Italy). This device is able to stimulate the multiplanar movements of lower limbs by combining various types of movement according to the physical principles of human motion. Working simultaneously on three axes by a sliding movement of lower limbs should activate different muscle groups following four positions at different intensity level. Muscles activity and training effectiveness were evaluated by monitoring Electromiography signal, metabolic data, oxygen uptake and heart rate. The goal of this research is to develop a system able to manage different information coming by varius electronic devices.

Keywords: EMG, Cardiofitness, Wavelet.

1 Introduction

More and more people are deciding to start exercising in a fitness center in order to increase muscle strength and reduce body fat. Several models of cardiofitness machines are available to practitioners who freely choose the exercise modality. However, incorrect exercise intensities and positions are related to inefficacious training and incidence of injuries 1..

Cardio Wave^{TM} (CW) is a cardiofitness apparatus specifically designed to allow movements along three axes (Fig. 1) by combining different movements which extensively train lower limb muscles according to four positions:

- Basic (B): with chest erect.
- Intermediate (I): with chest leant forward.
- Advanced (A): with chest and hands leant forward the ground.
- Free style (F): with arms free to oscillate.

Each position is meant to train different muscle groups.

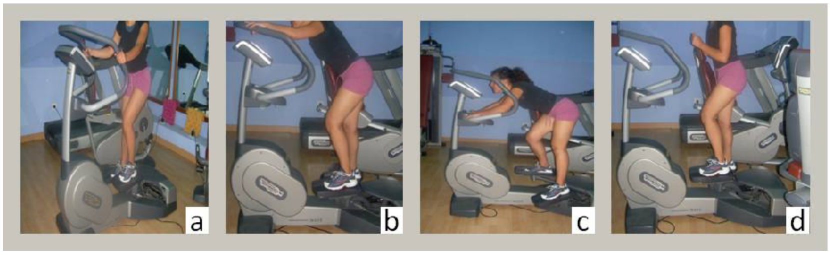


Fig. 1. Cardio Wave device positions: a) Standard , b) Advanced, c) Intermediate, d) Freestyle

During whatever sport activity fatigue is indicated by a decrease of muscle strength and power. Most studies of neuromuscular activity and fatigue have evaluated isometric and cyclical contractions. However, these types of contractions may not be representative of muscle activity and fatigue during physical exercise 2.. Indeed, available data suggest that the development of fatigue is specific to contraction type, intensity and duration of activity 3.. An exercise of the type that occurs during a fitness training session is characterized by a variety of muscle activities.

In order to investigate lower limb muscle activity and training optimization at the four positions (i.e. B, I, A, F) and in the 3 intensity levels, several parameters were monitored. Particularly Electromyography (EMG), Hearth Rate (HR), Oxygen uptake (VO2) and Lactate threshold (LA) were recorded by means of four different devices.

EMG was used to display muscle activation patterns and interpret both dysfunction and functional muscle recruitment 4.. HR is an important parameter to evaluate a specific exercise task in terms of the strain it places on the individual's aerobic system 5.. Both LA threshold and VO2 are predictors of aerobic exercise capacity 6..

Aim of this research was to evaluate training activities using the information coming from different electronic devices.

2 Methods

2.1 Subjects

During a test session nine healthy subjects - 5 trained and 4 untrained - with no history of musculoskeletal injuries and free from cardiac or metabolic disorders were recruited. They agreed to participate in the study. All subjects were selected on a voluntary basis from a fitness center population. They represented a wide spectrum of body weight, height, age and muscle strength. They ranged in age between 25 and 33 years with a mean value of 27.4 years. Their weight ranged from 65.4 kg to 99.8 kg with a mean value of 75.9 kg. Their height ranged from 167 cm to 177.8 cm with a mean value of 171.5 cm.

We considered as trained group the subjects that claimed to train at least twice a week at a moderate intensity (60-70% HR_{max}), for a minimum duration of 45 minutes / session.

Before training sessions all groups performed a 20-minute stretching protocol for lower limbs 7. for 30 seconds. This exercise was repeated 3 times for each muscle group. After a 10-minute warming-up on the ergometer, subjects were asked to perform randomly in all the four positions allowed by CW.

2.2 Data Recording

Commercial electronic devices are used to monitor different physics parameters for subject recruited and for each training session (Table 1).

Vastus medialis (VM), rectus femoris (RF) and biceps femoris (BF) muscles were selected for EMG analysis, as they are the most stimulated muscles in this kind of exercise.

The subjects were prepared for placement of EMG electrodes: the skin of each electrode site was shaved and cleaned carefully with alcohol in order to have a low inter-electrode resistance. The activity of each muscle was recorded by means of a couple of electrodes (Ag-Ag/Cl; Aurion s.r.l., Milan, Italy; 20 mm diameter, 20 mm inter-electrode distance). The electrodes were placed longitudinally on the motor point areas of the muscle examined. EMG signals were recorded telemetrically and the electrodes were placed on the RF, mid-way between the anterior superior iliac spine and the superior border of the patella; the BF over the long head, half-way between the ischial tuberosity and lateral femoral epicondyle; the VM at 80% on the line between the anterior spina iliaca and the joint space in front of the anterior border of the medial ligament 8..

To reduce movement artefacts, the electrodes were taped to the skin with an elastic bandage. Moreover, the usage of a wireless device allowed not to interfere with the subject's movements. EMG activity was acquired by using a wireless low power signal conditioning electronics device (ZeroWire Aurion s.r.l., Milan, Italy), in order to achieve both a stable input impedance and a high value of signal amplification (up to 100,000 V/V); a 2 KHz sample frequency was used for analog- to -digital conversion.

VO_2, HR and La concentrations were measured at rest and during exercise and recovery 9, 10..

During the test session, VO_2 was monitored continuously using a smaller portable gas analyzer (FitmatePro^{TM}, Cosmed, Rome, Italy), in order to assess cardio respiratory function. This device allowed the direct measurement of $VO_{2\ max}$ during an up to maximal exertion incremental exercise protocol, the extrapolation of $VO_{2\ max}$ during a sub-maximal incremental exercise, and the Estimation of $VO_{2\ max}$ based on the results achieved during some standard field tests.

Heart Rate was monitored by telemetric heart-rate bands (Polar, Kempele, Finland) with the registration of values during each minute of exercise on CW. We considered the last minute for statistical purposes 11-13.. The 4.0 mmol/l La threshold was measured with a lactate-meter (Accu-check, Roche Diagnostic, France) at rest and during exercise (at the end of each 3-minutes step) and recovery (3, 6, 9 minutes after performing the test).

All recording data coming from different devices were independently collected , were stored in a common database and processed togheter with a custom-written software in order to obtain information useful to a training activity analysis. Data coming from different devices use a storage memory of about 300 MegaByte for any subject. The computational time necessary to complete one test session is about 2 hours for any participant.

Table 1. Evaluated parameters during exercise

Time (min:s)	Intensity (RPM)	Parameter recording			
00:00	120	EMG	VO_2		HR
04:00	200	EMG	VO_2	LA	HR
08:00	300	EMG	VO_2	LA	HR
12:00	STOP			LA	HR
15.00	STOP			LA	
18:00	STOP			LA	
21:00	STOP			LA	

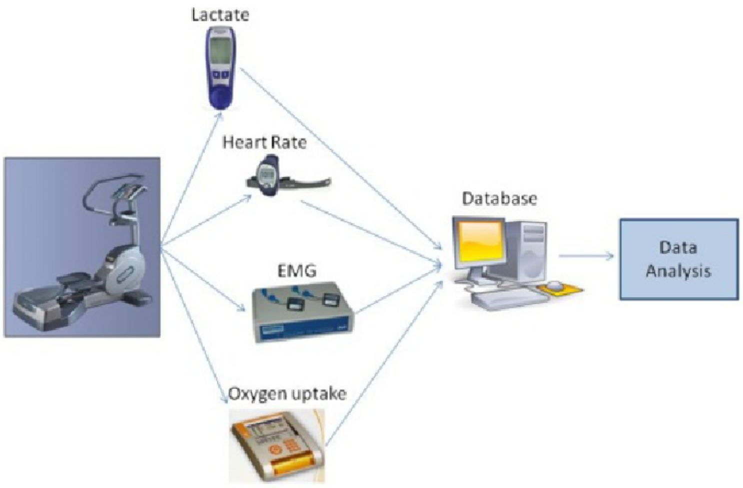


Fig. 2. Data flow

2.3 Fatiguing Protocol

We used a fitness-specific intermittent exercise protocol, which provided a fatiguing activity. Each subject performed 4 test sessions, one for each CW position (i.e., B, I, A, F). During a test session, subjects randomly performed the same 12-minute exercise at different intensities (4-minute step at three different intensities: low - 120 RPM, middle - 200 RPM and high intensity -300 RPM) imposed by the CW machine in the same position. In each session, subjects warmed up by 4 min/step at the three intensities selected; each session was separated by 180 min of recovery, but the subjects could only make 2 sessions per day in order to aid muscle fatigue recovery.

2.4 Data Processing

In order to test the different parameters for the purpose of evaluating muscle activity and fatigue, EMG data were analyzed by using LabView 8.0 software (National Instruments corp., Austin, TX, USA).

Continuous Wavelet Transform (CWT) was used in order to identify particular time-frequency signal models. Wavelet analysis is becoming a common tool for studying the localized variations of power within a time series. By decomposing a time series into time–frequency space, it is possible to evaluate both the dominant modes of variability and how those modes vary in time 14, 15..

In particular, in order to analyze EMG signals, Mean Instantaneous Frequency (MIF) was calculated. Starting from the spectrographic analysis of the signal, we aimed at pointing out how the signal's central frequency changes in time. MIF represents the power of spectral response for each subject studied. MIF is a mean value calculated to get a group parameter aimed at estimating how much the strength of muscle contraction and the degree of muscle fatigue vary in time 15.. MIF represents power spectral response for each subject trial: from MIF vector a mean value is calculated in order to obtain a clustering parameter.

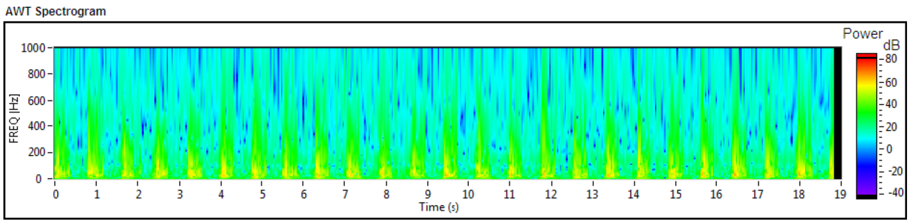


Fig. 3. EMG Wavelet Spectrogram

As for HR, LA and VO2 measures, once the data coming from the test were stored, the data themselves were processed by means of statistical tools. The processing was made in order to evaluate the change of each parameter according to the different positions and intensity levels.

In particular, for the EMG data, for each muscle a 4 (Positions) x 3 (intensity level) x 2 (Groups) ANOVA for repeated measures was applied to verify statistical differences in EMG data expressed as percentages of basal values. After data processing it is useful to analyse correlation between data coming from different devices.

3 Results

The data resulting from each test use a lot of storage memory and present a variety of typologies since they are the results of a processing of various signals. As a consequence, computational time to get the information about the quality of a training session is long.

MIF for each single muscle in the three intensity levels in both groups was computed in order to evaluate contraction power variation for each of them, thus underlining fatigue level. In tables 2 and 3 mean values of MIF in the four positions investigated for the different intensities are shown. From EMG's statistical analysis, no difference emerged between groups.

Table 2. MIF Values for T group at different positions and intensity levels

	Muscle	VM			RF			BF		
	Intensity	*120*	*200*	*300*	*120*	*200*	*300*	*120*	*200*	*300*
Positions	*B*	53.2	46.6	46.7	77.5	52.3	49.7	70.2	51.2	45.8
	I	92.3	78.6	79.1	98.5	79.7	80.1	70.3	78.0	79.3
	A	72.3	53.4	51.6	90.2	77.8	60.2	90.6	74.3	58.1
	F	80.6	61.3	59.9	78.8	60.1	61.2	92.4	70.3	71.2

Table 3. MIF Values for UT goup at different positions and intensity levels

	Muscle	VM			RF			BF		
	Intensity	*120*	*200*	*300*	*120*	*200*	*300*	*120*	*200*	*300*
Positions	*B*	70.5	60.2	60.1	100.1	82.2	77.6	130.1	82.0	80.0
	I	82.3	73.5	72.9	99.6	81.0	80.0	82.6	78.6	70.5
	A	78.6	66.4	67.0	99.8	96.5	83.4	120.8	102.3	99.0
	F	60.0	47.6	51.3	71.2	48.3	54.6	98.6	120.0	140.1

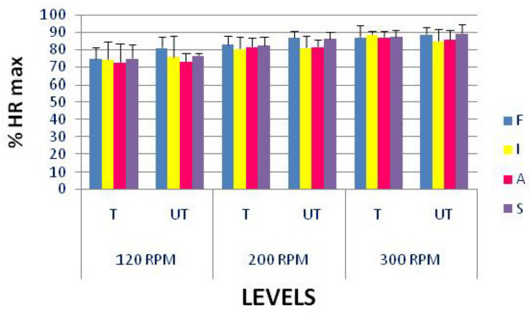


Fig. 4. Maximum Heart Rate value at different positions and intensity level for T and UT groups

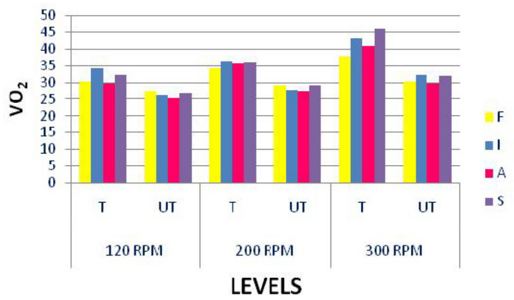


Fig. 5. VO2 mean value at different positions and intensity level for T and UT groups

About HR and VO_2 we observed a statistically significant difference only during the position F at the middle intensity among trained and untrained subjects, but the post hoc analysis did not show any significant difference (figure 4 and 5).

We did not find any significant difference for the blood lactate values in both groups.

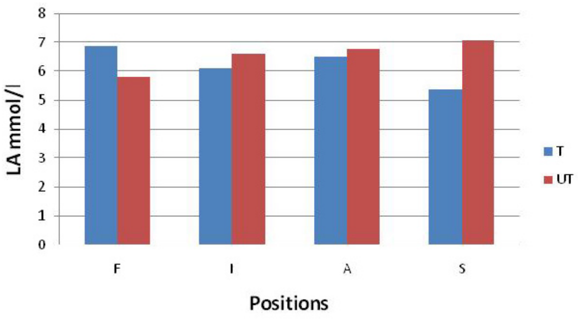


Fig. 6. Mean value of lactate peak at different positions and intensity level for T and UT groups

3 Discussion

On the basis of the results obtained, indications useful for the optimization of the training sessions were found. For instance, we observed an increase of the exercise intensity and muscular activation during the training session in F position. In UT group there was a trend towards a higher value of MIF, probably due to the lower economy of this intense exercise. However, the lack of statistical significance between T and UT may be attributed to the lower number of subjects.

As above mentioned, the insufficient statistical relevance of the data can also depend on the inefficiency of the whole measuring and processing procedures. These difficulties lengthen measuring and processing times a lot, thus making the monitoring of a high number of subjects difficult.

This problem might be solved by using a structure based on computer grids. In fact, nowadays this technology is very used for similar applications in different research fields 16, 17. (figure 7). Generally speaking, grid computing is the application of several computers to a single problem at the same time. It is usually about scientific or technical problems that require a great number of computer processing cycles or access to large amounts of data.

Ideally, a grid should provide full-scale integration of heterogeneus computing resources of any type: processing units, storage units, communication units, and so on 18.. For this problem a data grid can be used. It has the data storage capacity as its main shared resource. Such a grid can be regarded as a massive data storage system built up from portions of a large number of storage devices. Particularly, a remote user (e.g. a researcher) can manage data coming from different fitness centres and process them by using the grid resources at the same time, which would allow to manage also a much bigger database. Grid could be seen as a way to store data and to

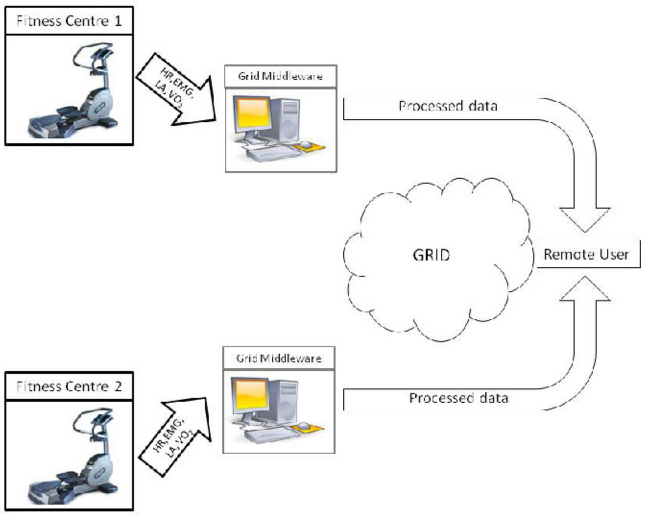


Fig. 7. Scheme of the proposed system on the grid

make them available to multiple users. The computing resources must satisfy basic requirements 18.: They must have enough computing power and data storage capacity to properly run the grid platform; they do not need to be directly connected to each other but they must know some entity that takes it to the grid; they can be indirectly connected through routers, gateways, hubs, switches, bridges, and wireless connections, by which a data packet can be sent from one computing resource to another.

For a data grid, the overall data storage capacity of a grid is the sum of the storage capacity made available for the grid in all its nodes and the performance of a data grid heavily depends on its communication links.

4 Conclusion

The paper presented a integrated system to manage and analyze different data in order to evaluate the training in cardiofitness activity. EMG, LA, HR and VO_2 were monitored during a training session and stored in a local database. The data were processed and a statistical analysis was performed in order to give usefully information to optimize the training session. Moreover a database containing all the monitored parameters for the different subjects was made. Future work will deal with the application of this system on the grid in order to improve its efficiency, reducing computational times and making the monitoring of a high number of subjects easy.

References

1. Siscovick, D.S., Weiss, N.S., Fletcher, R.H., Schoenbach, V.J., et al.: Habitual vigorous exercise and primary cardiac arrest: effect of other risk factors on the relationship. J. Chronic. Dis. 37(8), 625–631 (1984)

2. Rahnama, N., Reilly, T., Lees, A., Graham-Smith, P.: Muscle fatigue induced by exercise simulating the work rate of competitive soccer. J. Sports Sci. 21(11), 933–942 (2003)
3. Enoka, R.M., Stuart, D.G.: Neurobiology of muscle fatigue. J. Appl. Physiol. 72(5), 1631–1648 (1992)
4. Youdas, J.W., Hollman, J.H., Hitchcock, J.R., Hoyme, G.J., et al.: Comparison of hamstring and quadriceps femoris electromyographic activity between men and women during a single-limb squat on both a stable and labile surface. J. Strength Cond. Res. 21(1), 105–111 (2007)
5. Johnson, J.H., Prins, A.: Prediction of maximal heart rate during a submaximal work test. J. Sports Med. Phys. Fitness 31(1), 44–47 (1991)
6. Kawabata, T., Suzuki, T., Miyagawa, T.: Effect of blood volume on plasma volume shift during exercise. Journal of Thermal Biology 29(7-8), 775–778 (2004)
7. Bandy, W.D., Irion, J.M., Briggler, M.: The effect of static stretch and dynamic range of motion training on the flexibility of the hamstring muscles. J. Orthop. Sports Phys. Ther. 27(4), 295–300 (1998)
8. Hermens, H.J., Freriks, B.: The State of the Art on Sensors and Sensor Placement Procedures for Surface ElectroMyoGraphy: A proposal for sensor placement procedures, Enschede - The Netherlands: Roessingh Research and Development (1997)
9. Kang, J., Chaloupka, E.C., Mastrangelo, M.A., Hoffman, J.R., et al.: Metabolic and perceptual responses during Spinning cycle exercise. Med. Sci. Sports Exerc. 37(5), 853–859 (2005)
10. Green, J.M., McLester, J.R., Crews, T.R., Wickwire, P.J., et al.: RPE-lactate dissociation during extended cycling. Eur. J. Appl. Physiol. 94(1-2), 145–150 (2005)
11. Kang, J., Mangine, G.T., Ratamess, N.A., Faigenbaum, A.D., et al.: Influence of intensity fluctuation on exercise metabolism. Eur. J. Appl. Physiol. 100(3), 253–260 (2007)
12. Green, J.M., McLester, J.R., Crews, T.R., Wickwire, P.J., et al.: RPE association with lactate and heart rate during high-intensity interval cycling. Med. Sci. Sports Exerc. 38(1), 167–172 (2006)
13. Dolbow, D.R., Farley, R.S., Kim, J.K., Caputo, J.L.: Oxygen consumption, heart rate, rating of perceived exertion, and systolic blood pressure with water treadmill walking. J. Aging. Phys. Act. 16(1), 14–23 (2008)
14. Kumar, D.K., Pah, N.D., Bradley, A.: Wavelet analysis of surface electromyography to determine muscle fatigue. IEEE Trans. Neural Syst. Rehabil. Eng. 11(4), 400–406 (2003)
15. Ren, X., Yan, Z., Wang, Z., Hu, X.: Noise reduction based on ICA decomposition and wavelet transform for the extraction of motor unit action potentials. J. Neurosci. Methods 158(2), 313–322 (2006)
16. Frizziero, E., Gulmini, M., Lelli, F., Maron, G., et al.: Instrument Element: a new Grid component that enables the control of remote instrumentation. In: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW 2006). IEEE, Los Alamitos (2006)
17. Lelli, F., Frizziero, E., Gulmini, M., Maron, G., et al.: The many Faces of the Integration of Instruments and the Grid. International Journal of Web and Grid Services 3(3), 239–266 (2007)
18. Costa, S.R.R., Neves, L.G., Ayres, F., Mendonca, C.E., et al.: GridBR: The challenge of grid computing. Grid and Cooperative Computing, pt. 1, 601–607 (2004)

Gene Specific Co-regulation Discovery: An Improved Approach[⋆]

Ji Zhang, Qing Liu, and Kai Xu

CSIRO Tasmanian ICT Centre
Hobart, TAS, Australia 7001
{Ji.Zhang,Q.Liu,Kai.Xu}@csiro.au

Abstract. Discovering gene co-regulatory relationships is a new but important research problem in DNA microarray data analysis. The problem of gene specific co-regulation discovery is to, for a particular gene of interest, called the *target gene*, identify its strongly co-regulated genes and the condition subsets where such strong gene co-regulations are observed. The study on this problem can contribute to a better understanding and characterization of the target gene. The existing method, using the genetic algorithm (GA), is slow due to its expensive fitness evaluation and long individual representation. In this paper, we propose an improved method for finding gene specific co-regulations. Compared with the current method, our method features a notably improved efficiency. We employ kNN Search Table to substantially speed up fitness evaluation in the GA. We also propose a more compact representation scheme for encoding individuals in the GA, which contributes to faster crossover and mutation operations. Experimental results with a real-life gene microarray data set demonstrate the improved efficiency of our technique compared with the current method.

1 Introduction

DNA microarray is an enabling technology to provide a global view of the expression of a large number of genes. A gene microarray data set is typically presented as matrix where each row represents a gene and each column is the experimental condition (such as time point) when the gene expression is extracted. Finding gene co-regulatory relationships is an important research focus in microarray data analysis. One interesting research problem, called *Single Gene Approach* for gene microarray analysis [14], was recently studied by [17]. This problem can be formulated as follows: *for a particular gene of interest, called target gene, identify its strongly co-regulated genes and the condition subsets where such strong gene co-regulations are observed.* The discovered co-regulated genes and the associated condition subsets are specific to the target gene, which can help biologists to better understand and characterize it. This is useful in many

⋆ This research is supported by CSIRO Preventative-Health Flagship program and Tasmanian ICT Centre (TasICTC). TasICTC is jointly funded by the Australian Government through the Intelligent Island Program (administered by the Tasmanian Department of Economic Development and Tourism) and CSIRO.

applications such as the investigation of the most differentially expressed genes in a disease study or the function prediction of unknown genes.

This paper proposes two boosting techniques aiming to achieve a noticeable efficiency improvement for the method proposed in [17]. The major technical contributions of this paper are summarized as follows:

1. First, we employ kNN Search Table [16] to find the lower and upper bounds of the distance between the target gene and its k^{th} most co-regulated genes in the data set. These bounds can be utilized to substantially speed up fitness evaluation in the GA and facilitate the procurement of the top condition subsets where the target gene is most co-regulated with other genes in the microarray data set. We have also devised a better way to specify the parameter values that are used in the approximation scheme;
2. We also propose a more compact representation scheme for encoding condition subsets in the GA, which contributes faster crossover and mutation operations.
3. Experimental results with a real-life gene microarray data demonstrate the better efficiency of our technique than that of the existing method in discovering gene specific co-regulations.

The remainder of this paper is organized as follows. Section 2 presents the approach to efficiently find the top co-regulated condition subsets for the target gene using kNN Search Table. A more compact individual representation scheme is proposed in Section 3 to speedup the GA by reducing the overhead of crossover and mutation operations. We report the experimental results in Section 4 and the last section concludes this paper.

2 Related Work

Clustering analysis is probably the most commonly used technique for studying gene co-regulations by grouping closely co-regulated genes together. The major clustering algorithms in discovering gene co-regulations include hierarchical clustering method [11], k-means algorithm [11], Self-Organization Maps (SOMs) [12] and SVD-based clustering algorithm [8]. Because they perform clustering based on the entire set of conditions (*i.e.*, full dimensionality), thus they miss out those interesting co-regulations embedded in the lower dimensional condition subsets. To find the gene co-regulations in some subsets of conditions, a few subspace clustering methods for gene expression data, such as Coupled Two-Way Clustering [7], bi-cluster [3] and δ-cluster [15], are also proposed. However, a common key drawback for clustering methods, no matter whether they find co-regulations on the full or partial dimensionality, is that there is no guarantee that the target gene's most-co-regulated genes are in the same cluster where the target gene is located. Quite likely, they are located in a few different clusters because the results of clustering are quite sensitive to the parameters such as the number of clusters to be obtained or the value of the inter-cluster dis-similarity metric that users choose. In addition, the clustering analysis needs to be performed in a large number of condition subsets, which is rather expensive. As such, clustering is not a direct nor efficient way for discovering gene-specific co-regulations.

There are also some work on gene co-regulation discovery from time series perspective by considering the explicit temporal nature of the features in the gene microarray data [1][2][4][5][6]. However, there are cases that the features do not have explicit temporal meaning and gene co-regulations may occur in the subsets of features that are far apart from each other in the gene microarray data set.

In [17], the authors proposed an approach for mining local gene-specific co-regulation using genetic algorithm (GA) [9]. The basic idea of this approach is to first find the condition subsets in which the target gene g is most significantly co-regulated with others and the co-regulated genes of g are then selected from its nearest neighbors in these condition subsets. A sliding window is used to scan all the conditions sequentially and the search of condition subsets is performed within each window position. This method is able to find the closely co-regulated genes for the target gene and the associated condition subsets where such co-regulation occur. However, this method is slow. The major speed bottleneck of this approach is the fitness computation in the GA, which involves a kNN search for the target gene in each condition subset. The typically large number of genes in the microarray data and the number of condition subsets evaluated in the GA lead to a slow kNN search.

3 Searching Co-regulated Condition Subsets Using *k*NN Search Table

In [17], the fitness function of the target gene in each condition subset is defined as the distance between it and its k^{th} nearest gene in the microarray data set. Such kNN search has to be performed in all the subspaces that are explored by the GA, leading to a slow speed for the whole method. Using indexing methods to speed up k^{th}NN search is not efficient due to two major reasons. First, since a large number of condition subsets may be evaluated in the GA, it will be expensive to index the genes in each possible condition subset. Second, for gene-specific co-regulation discovery problem, we may be only interested in a small number of target genes, thus the high cost associated in building indexing cannot be amortized by the one-time performance gain by using the indexing.To solve this problem, we draw on the *kNN Search Table* proposed in [16] to speed up computation of fitness function. A kNN Search Table for a target gene g, denoted as $\mathcal{T}^g$, is a $M \times k$ table containing its k nearest neighbors in *each* single dimension of full data space with M dimensions. The entry x_{ij} of the table represents the j^{th} nearest neighbor of g in the i^{th} dimension, where $1 \leq i \leq M$ and $1 \leq j \leq k$. Using kNN Search Table, we can compute very efficiently the lower and upper bounds of the distance between the target gene and its k^{th} nearest neighbor in any condition subset s, denoted by $f^k_{min}(g, s)$ and $f^k_{max}(g, s)$, respectively. Interested readers can refer to [16] for the computation of $f^k_{min}(g, s)$ and $f^k_{max}(g, s)$ using kNN Search Table and the proof of their correctness.

The idea of using kNN Search Table for speeding up the fitness computation of condition subset s with respect to the target gene g is to approximate the fitness by using the linear combination of $f^k_{min}(g, s)$ and $f^k_{max}(g, s)$ as follows:

$$fitness_{app}(g, s) = \alpha f^k_{min}(g, s) + \beta f^k_{max}(g, s)$$

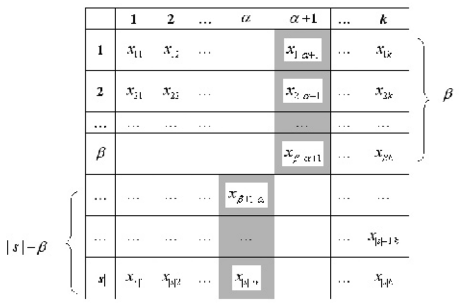


Fig. 1. kNN Search Table

where $\alpha + \beta = 1$. Unlike setting $\alpha = \beta = \frac{1}{2}$ as in [16], we calculate the accurate fitness, f^k_{min} and f^k_{max} for a specified number of condition subsets in the GA to set the values for α and β for each condition subset s such that the following equation is satisfied:

$$fitness_{accurate}(g, s) = \alpha f^k_{min}(g, s) + \beta f^k_{max}(g, s)$$

subject to $\alpha + \beta = 1$. The average of α and β of the test condition subsets are computed and used for the fast fitness approximation of all the other condition subsets in the GA.

kNN Search Table is advantageous in the following two aspects: 1) The construction of kNN Search Table w.r.t. the target gene only involves finding its kNNs in each one-dimensional condition subset, therefore its construction cost is only $\mathcal{O}(NM)$, which is linear with respect to both number and dimensionality of genes in the gene microarray data set. For the whole gene microarray data, only one kNN Search Table needs to be pre-computed for a target gene; 2) The total complexity for computing $f^k_{min}(g, s)$ and $f^k_{max}(g, s)$ is $O(k^2|s|^2)$, which becomes independent of N and M.

Since $fitness(g, s)$ is approximated in the GA, the accuracy of computation is thus somehow affected. To address this problem, we can perform a *refinement step* on the top condition subsets we obtain in the GA that are stored in the so-called *CandidateSet*. Instead of using f^k_{min} and f^k_{max} for a fast fitness approximation, the refinement step computes the accurate fitness for top candidate condition subsets and the top-k condition subsets among them will be returned. Admittedly, evaluating each condition subset in this refinement process is more expensive than the approximation as it involves more accurate computations. However, the number of candidate condition subsets is typically much smaller than the total number of condition subsets approximated in the GA. In addition, a pruning optimization strategy can be further devised to speed up the computation, which operates as follows. After the fitness of n candidate condition subsets have been accurately evaluated, we start to maintain the minimum $fitness(g, s)$ for the top-k condition subsets we have found thus far, denoted as $MinFit$. Those unevaluated condition subsets that satisfies $f^k_{max}(g, s) < MinFit$ cannot become the top-k condition subsets and can therefore be safely pruned. This is because that $MinFit$ is monotonically increasing as we examine more condition subsets in the refinement step.

Once the top-ranked condition subsets have been found by the GA, the next step will be finding the co-regulated condition subsets from these top-ranked condition subsets. This step will be much trivial than the step of finding the top co-regulated condition subsets. It only involves finding the k most co-regulated genes of the target gene in each of those top-ranked condition subsets.

4 Shorter Individual Representation

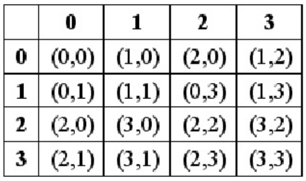
A straightforward representation scheme for condition subsets in the GA is the standard binary encoding; all individuals are represented by strings with fixed and equal length M, where M is the number of dimensions of the Microarry data set. Using binary alphabet $\Sigma = \{0, 1\}$ for gene alleles, each bit in the individual will take on the value of "0" and "1", respectively, indicating whether or not its corresponding attribute is selected

However, for a high-dimensional microarray data set with M dimensions, we will end up with a long binary string with a length of M for representing each condition subset. A high computational overhead is involved in the frequently preformed crossover and mutation of long binary strings. Therefore, a short representation of individuals is desirable. To this end, we employ integer string, instead of binary string, to represent each condition subset in the GA that features a much shorter length. Each integer can represent a few binary bits, which contributes a remarkable reduction of the length of individual representation. If we assume that the number of binary bits needed for representing each integer is L, $L \leq M$ (integers are in the range of $[0, 2^L - 1]$), then length of an integer string for representing a condition subset will be only approximately $\frac{M}{L}$ of the length of the binary string used to represent the same condition subset.

Even though it features a more compact representation and enables faster crossover and mutation operations, the integer string representation changes the behavior of mutation and tends to result in abrupt mutations more frequently, which may lead to loss of useful segments of bits that might just turn out to be part of potentially good solutions.

To solve these problems while, at the same time, preserve the desired advantages of integer representation, we propose efficient methods to seamlessly simulate the genetic operations of binary string using integer string representation. We derive ways for quickly obtaining crossover and mutation results between any pair of integers that are consistent with those of binary strings.

For crossover operations, we propose *Crossover Lookup Table (CLT).* It is a $2^L \times 2^L$ table with each entry being a pair of integers corresponding to the crossover result of a given pair of integers. The crossover locus l_c is generated randomly in the range of $[1, M-1]$. Notice that, from an integer string's perspective, the crossover locus will only be located on the boundary of two adjacent integers or within a single integer. In the former case, two integer strings can be directly crossovered in the same way as the binary strings. In the later case, all the integers after the one that the crossover locus is located can also be crossovered in the same way as the binary strings. The crossover result of the



	0	1	2	3
0	(0,0)	(1,0)	(2,0)	(1,2)
1	(0,1)	(1,1)	(0,3)	(1,3)
2	(2,0)	(3,0)	(2,2)	(3,2)
3	(2,1)	(3,1)	(2,3)	(3,3)

Fig. 2. An example of Crossover Lookup Table ($identifier = 1$, $L = 2$)

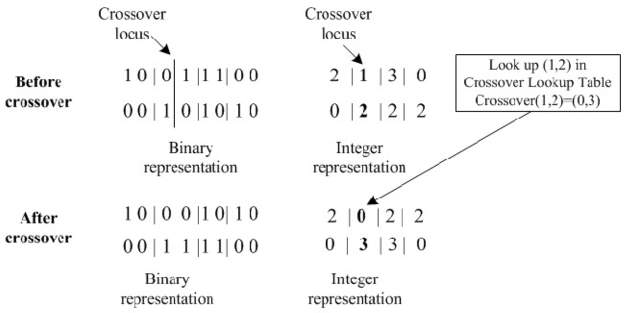


Fig. 3. A crossover example of two integer strings ($M = 8$, $L = 2$, $l_c = 3$)

pair of integers where crossover locus is located can be obtained by looking up the *appropriate* Crossover lookup Table based on the value of (l_c mod L). As there are $L - 1$ different crossover locus inside an integer, thus we need to pre-computed $L - 1$ different CLTs. Each table is uniquely identified by an integer $i \in [1, L - 1]$. Figure 2 gives an example CLT with identifier=1 when $L = 2$. Figure 3 is a crossover example of two integers by means of the CLT given in Figure 2.

For mutation operations, we quantify the *Mutation Transition Probability Table (MTPT)*. It is also a $2^L \times 2^L$ table with each entry representing the mutation transition probability from one integer to another. An integer is mutated to another based upon their transition probability initiated from this integer. Let us suppose that two integers a and b differ in l bites in their binary representations ($0 \leq l \leq L \leq M$), the mutation transition probability from a to b, denoted as $Pr(a, b)$, is computed as $Pr(a, b) = p_m^l(1 - p_m)^{(L-l)}$. Unlike CLT, there is only one MTPT that needs to be pre-computed for a given gene microarray data set. Suppose $L = 2$ and $p_m = 0.1$, we need to compute the mutation transition probability from integer 2 to 3. As they differ in only 1 bit in their binary representations, thus $Pr(2, 3)$ can be computed as $Pr(2, 3) = 0.1^1 \cdot (1-0.1)^{2-1} = 0.09$.

When using a shorter integer representation for condition subsets in the GA, we can achieve an approximate $\frac{M}{L}$ times performance boost in crossover and mutation operations for M-dimensional gene microarray data if each integer in the string is represented by L binary bits. The pre-computed CLT and MTPT contribute to a remarkable performance gain of crossover and mutation in the GA by transforming them to simple lookup operations from lookup tables, without the frequent on-the-fly conversion between integer and binary strings.

5 Experimental Results

The Spellman's data set is used in our experiments that can be downloaded from http://genome-www.stanford.edu/cellcycle/data/rawdata. This data set contains all the data for the alpha factor, cdc15, and elutriation time courses, and includes the data for the Clb2 and Cln3 induction experiments. We used only the alpha-factor and CDC28 data set for our experiments. This data set contains 6178 genes under 35 experimental conditions (time points).

The main focus of our experimental evaluation is to investigate the efficiency boost of the existing method when the two strategies we propose (*i.e.*, fitness evaluation using kNN Search Table and shorter individual representation scheme using integer strings) are employed. Efficiency comparison is conducted in the experiment between the baseline method (*i.e.*, no boosting strategies are used) and the method with only one boosting strategy is used. In this way, we can obtain a clear idea as to the magnitude of improvement that can be achieved using each strategy. For all the experiments, the number of generations and population size are fixed as 200 and 50 respectively in the GA. The efficiency are investigated under varying number of genes and conditions. The original gene microarray data is properly sampled to obtain new data sets with desired number of genes and conditions for experimental purposes. At the end of this section, we will also investigate the effectiveness of our proposed improved method.

5.1 Efficiency Improvement Using kNN Search Table

We first study the efficiency boost of the method using kNN Search Table. Figure 4 and 5 show the execution time of the method using kNN Search Table, compared with the baseline method. By using kNN Search Table, the method can achieve a remarkable improvement of efficiency, especially when the number of genes and conditions of the microarray data set are high. The magnitude of speed improvement could be over 70% for the microarray data we use in the experiment. As we have analyzed earlier, the complexity of using kNN Search Table to approximate the fitness of condition subsets in the GA is independent of the number of genes and the number of condition of the microarray data set, thus approximately horizontal curves for the boosted method are observed in both figures. Note that the execution time of the improved method increases only slightly when dealing with data sets with higher number of genes or conditions. Such increase comes from the refinement step where the top co-regulated condition subsets for the target gene are obtained based on the top-ranked condition subsets returned by the GA. Insteading of using kNN Search Table, this step involves evaluating all the genes in the data set whose complexity is depended on the number of genes and conditions.

5.2 Efficiency Improvement Using Shorter Representation

The second experiment investigates the efficiency improvement of the method by using integer strings for representing condition subsets in the GA. The experimental results are presented in Figure 6 and 7, respectively. As these results show, employing integer string representation is able to contribute to another

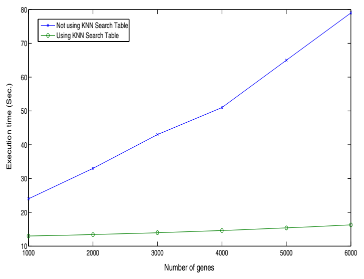


Fig. 4. Efficiency improvement using kNN Search Table (under varying number of genes)

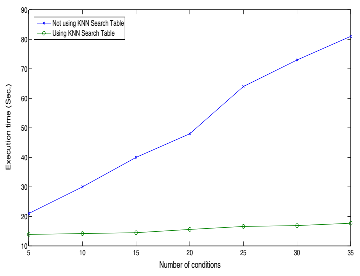


Fig. 5. Efficiency improvement using kNN Search Table (under varying number of conditions)

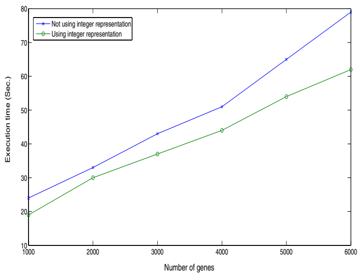


Fig. 6. Efficiency improvement using integer representation (under varying number of genes)

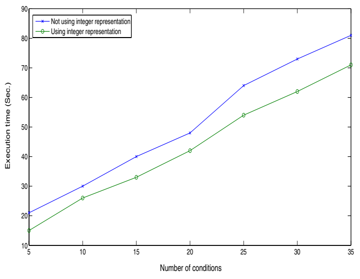


Fig. 7. Efficiency improvement using integer representation (under varying number of conditions)

10-15% speed improvement. Also, as the number of genes or conditions increase, the magnitude of speed inprovement using integer representation is somewhat decrease. This is because that the cost of fitness function dominates the whole algorithm more severely when the number of genes or conditions increase, making the efficiency contirubtion of integer representation relatively less remarkable.

5.3 Effectiveness Study

Besides efficiency improvement, we are also interested in studying the effectiveness of the method in gene specific co-regulations discovery when the boosting strategies are used. A general rationale is that the effectiveness of the method should not be severely compromised when its speed can be improved noticeably. In this experiment, we study the final co-regulated condition subsets of the target gene obtained using the improved method, compared with the baseline method. As we know, as long as the condition subsets for gene co-regulation of the target gene can be correctly identified, it will then become trivial to find its co-regulated genes in these condition subsets. Therefore, we only need to

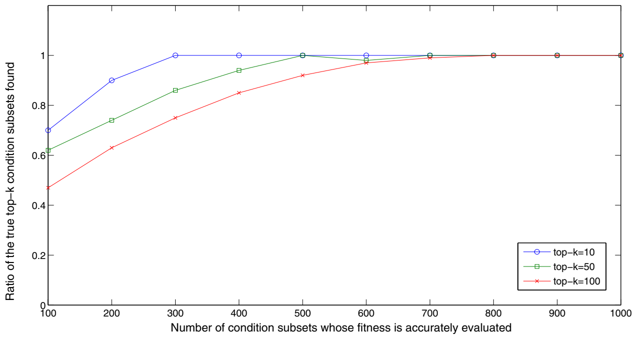


Fig. 8. Effectiveness of the improved method

evaluate how good are the final co-regulated condition subsets obtained by the GA. The ground-truth result, *i.e.*, the true top-k condition subsets where the strongest co-regulation for the target gene, needs to be found first using the brute-force search method. Based on the ground-truth result, the result of the improved method can be evaluated.

In this experiment, we vary the number of top condition subsets to be returned by the GA, denoted by N_s, and study the percentage of the true top-k that are found using the boosting strategies under three top-k values, 10, 50 and 100. Intuitively, the larger N_s is, the higher chance will be for the final results of the GA to include the true top-k condition subsets. As approximation is used in the GA, N_s should be set relatively large (yet still much smaller than the total number of condition subsets that are evaluated in the GA) in order to provide a sufficiently large room to include the true top-k condition subsets. We evaluate the different values for N_s, ranging from 100 to 1000, in this experiment. The results, as presented in Figure 8, show that we will be able to obtain all the true top-k condition subsets as long as the N_s is reasonably large (the required value for N_s is increased when top-k goes up). In other words, we only need to evaluate the fitness of a small number of condition subsets accurately to find the true top-k condition subsets, and the fitness of all the other condition subsets can be quickly approximated using kNN Search Table.

6 Conclusions

In this paper, we propose an improved method for discovering gene specific co-regulations based on genetic algorithm (GA). We employ kNN Search Table to substantially speed up fitness evaluation in the GA. We also propose a more compact representation scheme for encoding condition subsets in the GA, which contributes faster crossover and mutation operations. Our method features a better efficiency than the current method (up to a 70%-80% speed boost). The experimental results demonstrate the good efficiency and effectiveness of our technique in discovering gene specific co-regulations.

References

1. Amato, R., Ciaramella, A., Deniskina, N., Del Mondo, C., di Bernardo, D., Donalek, C., Longo, G., Mangano, G., Miele, G., Raiconi, G., Staiano, A., Tagliaferri, R.: A Multi-step Approach to Time Series Analysis and Gene Expression clustering. Bioinformatics 22(5), 589–596 (2006)
2. Bar-Joseph, Z.: Analyzing time series gene expression data. Bioinformatics 20(16), 2493–2503 (2004)
3. Cheng, Y., Church, G.M.: Biclustering of Expression Data. In: Proc. International Conference on Intelligent Systems for Molecular Biology (ISMB), vol. 8, pp. 93–103 (2000)
4. Erdal, S., Ozturk, O., Armbruster, D., Ferhatosmanoglu, H., Ray, W.: A time series analysis of microarray data. In: 4th IEEE International Symposium on Bioinformatics and Bioengineering (2004)
5. Feng, J., Barbano, P.E., Mishra, B.: Time-frequency feature detection for time-course microarray data. In: 2004 ACM Symposium on Applied Computing (SAC 2004) (2004)
6. Filkov, V., Skiena, S., Zhi, J.: Analysis techniques for microarray time-series data. In: 5th Annual International Conference on Computational Biology (2001)
7. Getz, G., Levine, E., Domany, E.: Coupled Two-Way Clustering Analysis of Gene Microarray Data. Proc. Natioal Academy of Science 97(22), 12079–12084 (2000)
8. Golub, G.H., Van Loan, C.F.: Matrix Computations. Johns Hopkins University Press (1983)
9. Holland, J.: Adaption in Natural and Artificial Systems. MIT Press, Cambridge (1992)
10. Ji, L., Tan, K.L.: Identifying Time-Lagged Gene Clusters on Gene Expression Data. Bioinformatics 21(4), 509–516 (2005)
11. Johnson, R.A., Wichern, D.W.: Applied Multivariate Statistical Analysis. Prentice Hall International, USA (1998)
12. Kohonen, T.: Self-Organization Maps. Springer, Heidelberg (1995)
13. Ramoni, M.F., Sebastiani, P., Kohane, I.S.: Cluster analysis of gene expression dynamics. Proceedings of the National Academy of Sciences, USA 99(14), 9121–9126 (2002)
14. Speed, T., Fridlyand, J., Yang, Y.H., Dudoit, S.: Discrimination and clustering with microarray gene expression data. In: Spring Meeting of International Biometric Society Eastern North American Region (ENAR 2001) (2001)
15. Yang, J., Wang, W., Wang, H., Yu, P.S.: δ-Cluster: Capturing Subspace Correlation in a Large Data Set. In: Proc. 18th International Conference on Data Engineering (ICDE 2002), pp. 517–528 (2002)
16. Zhang, J., Gao, Q., Wang, H.: A Novel Method for Detecting Outlying Subspaces in High-dimensional Databases Using Genetic Algorithm. In: Perner, P. (ed.) ICDM 2006. LNCS, vol. 4065, pp. 731–740. Springer, Heidelberg (2006)
17. Zhang, J., Gao, Q., Wang, H.: Discover Gene Specific Local Co-regulations Using Progressive Genetic Algorithm. In: ICTAI 2006, pp. 783–790 (2006)

Experimental Evaluation of Protein Secondary Structure Predictors

Luca Miceli[1], Luigi Palopoli[1], Simona E. Rombo[1], Giorgio Terracina[2], Giuseppe Tradigo[3], and Pierangelo Veltri[3]

[1] DEIS, Università della Calabria
[2] Dip. di Mat., Università della Calabria
[3] Dip. Med. Sp. e Cl., Univ. di Catanzaro

Abstract. Understanding protein biological function is a key issue in modern biology, which is largely determined by its 3D shape. Protein 3D shape, in its turn, is functionally implied by its amino acid sequence. Since the direct inspection of such 3D structures is rather expensive and time consuming, a number of software techniques have been developed in the last few years that predict a spatial model, either of the secondary or of the tertiary form, for a given target protein starting from its amino acid sequence.

This paper offers a comparison of several available automatic secondary structure prediction tools. The comparison is of the experimental kind, where two relevant sets of proteins, a non-redundant one including 100 elements, and a 180-protein set taken from the CASP 6 contest, were used as test cases. Comparisons have been based on evaluating standard quality measures, such as the Q3 and SOV.

1 Introduction

Proteins are the basic constituents of living beings. They form the basis for structural components of cells as well as for metabolic processes involved in organic life. Understanding protein functions has, therefore, a central role in the analysis of the biological mechanisms underlying life processes. A protein biological function is largely determined by its 3D shape [15], which is functionally implied, in its turn, by the sequence of amino acids that form the protein [3]. The amino acidic sequence of a protein is called its *primary structure*, whereas its 3D shape is encoded in two different representations, namely, its *secondary* and its *tertiary* structure [15]. To illustrate, the tertiary structure of a protein tells, with respect to a given 3D fixed axis origin point, the exact positions of protein constituent atoms. The secondary structure, instead, provides information about the composition of the protein structure in terms of regular substructures. In fact, amino acids tend to dispose themselves within some few substructures, namely, sheets consisting of β*-strands* (denoted E in the following) laterally connected to form a pleated sheet, and α*-helices* (denoted H in the following). Moreover, amino acids might contribute to form kinds of irregular structures, which link regular ones to one another, and which are usually referred to as *loops* (denoted L in the sequel). Thus, a secondary protein structure is denoted by a sequence of letters over the alphabet $E,\ H,\ L$, one letter for each of the amino acids occurring in the primary structure of the protein. These letters are called the *conformational states* of the amino acids.

All that given, the relevance of associating secondary and tertiary structure with protein amino acid sequences is immediately understood. To do that, complex lab methods (that are, X-ray crystallography and nucleic magnetic resonance spectroscopy (NMR)) can be employed, which are however expensive and time-consuming. As a result, protein sequence discovering rate is much higher than protein structure identification rate and, therefore, while some millions of protein primary structures are known to date, only a few tens of thousands of secondary and tertiary structures have been discovered [2]. This is the reason why a large interest has been witnessed in the community towards computational methods for *predicting* [10,25,26,28] such secondary and tertiary structures [4,5,13,16,17,21,22,29]. Prediction methods, though fast and cheap to run, do not achieve to date the same accuracy as lab methods in reconstructing protein structures.

This paper is concerned with comparing computational methods for protein *secondary* structure prediction. Therefore, in the following, we will make no further reference to tertiary structures. In this setting, this paper offers an experimental analysis of several available protein secondary structure predictors. The analysis has been conducted by using two relevant data sets (a non-redundant and a CASP protein set), which will be described below and overall including 280 proteins. The experimental evaluation has been carried out by querying a set of available prediction tools over proteins having known secondary structure, in order to evaluate their accuracy according to quality evaluation parameters, namely, the Q3 and the SOV, which are commonly adopted in the literature [23,27]. Both the average behavior of each predictor on the whole data sets and its specific accuracy for each protein are reported and analyzed so as to result in a comparison of the (relative) performances of the considered prediction tools.

It is worth mentioning here that monitoring tools like those published by *EVA* server [7] can be referred to in order to verify the quality of results delivered by prediction tools made available on-line, providing the evaluations $Q3$ and SOV for secondary structure prediction. In particular, at for today, the average values for such parameters are 75.9% for $Q3$ and 72.7% for SOV, according to data reported at `http://cubic.bioc.columbia.edu/eva/sec/res_sec.html`, where 100% would be the value scored by the *perfect* predictor for both parameters. International challenges, such as the biannual *CASP* competition [30], have been instituted to encourage studying and designing high-quality synthetic predictors, even if CASP focuses on 3D structure prediction tools.

The work presented in this paper differs in both objectives and contributions from those either resulting from the CASP competition or obtained using the EVA server, for the following reasons. First, we notice that CASP is intended to evaluate the impact of current prediction methods and techniques in helping experts to design 3D structure predictions; in fact, groups competing in CASP are not constrained to exploit automatic tools only; rather, they can refine "by hand" resulting predictions with the guide of current human expertise in the field. As a consequence, CASP is meant to determine a measure of human capabilities in predicting protein structures with the aid of available automatic tools, rather than on evaluating prediction tools "per se". So, CASP is, in a sense, *expert*-oriented. On the contrary, the purpose of EVA is to determine what confidence a biologist should rely on a given specific tool "as they are" to be a good predictor, by monitoring the quality of its predictions on a wide variety of proteins and a large time range. Therefore, EVA is intended to evaluate the performances of the single tools, where different protein data sets may be used in evaluating a tool, with no reference whatsoever to those which were used to evaluate *other* tools.

Therefore, neither CASP, nor EVA address direct experimental cross-comparison between prediction tools, which is the focus of this paper, inasmuch as the objective of our work is to provide a thorough comparison of tool performances on protein data sets specifically selected to be both statistically and biologically relevant.

The rest of the paper is organized as follows. The next section presents a very brief overview of main tool categories, whereas Section 3 introduces data sets and prediction tools we exploited for our experimental evaluations, illustrates the experimental results, and discusses about predictor performances as resulting from the experiments is reported. Finally, in Section 4, some conclusions are drawn.

2 Secondary Structure Prediction Methods

As already stated, secondary structure prediction consists in associating a string of characters representing amino acids conformational states to the string representing the primary structure of a protein sequence, whose structure is not determined experimentally. The correctness of the prediction can only be determined by comparing the predicted string with the secondary structure obtained from lab methods (called its *observed* structure), once it has been singled out [26]. It is largely accepted that evaluation of quality of a prediction is done by using $Q3$ and SOV parameters, where $Q3$ punctually measures the percentage of correctly guessed structures for the target protein, i.e., the conformational state of single amino acids, whereas SOV is obtained by computing per-segment overlaps [7].

Protein secondary structure prediction methods (as well as tertiary ones) can be classified as either *ab initio* methods or *evolution-based* methods (aka, *homology-based* methods). In the first case protein structure prediction is based on evaluating the minimal free energy [6]. Indeed the three dimensional conformation of a protein is defined by the spatial conformation of amino acids chain that correspond to that structure featuring the lowest free energy determined by the mutual interactions of amino acids. This is the idea underlying the development of *ab initio* methods. However, an exhaustive search of all possible configurations of a polypeptide chain is such a formidable task that ab initio methods usually produce satisfactory results just in case of chains with a low number of amino acids, in which cases, it is actually feasible to evaluate the energy configuration of the folding process rather precisely. Simulating algorithms (such as Monte Carlo methods) are useful in the case of proteins with a low number of amino acids, or in the case of prediction processes guided by information used to select (and, thus, reduce) the number of possible configurations to test for good. As an example, the Rosetta [24] predictor uses a Monte Carlo algorithm to reduce the possible combinations of amino acids while predicting single regions.

Counter-wisely, *evolution-based* methods look at the target protein's primary structure, comparing this protein sequence with known ones published in available databases and trying to exploit evolutionary protein relationships (e.g., with protein family identification). Then, comparative modeling algorithms are used to build a spatial configuration for the unknown structure. Given a set of proteins that are evolutionary related to the target one, comparative based algorithms construct a multi alignment and deduce correspondences between amino acids, possibly identify regions affected by insertion, deletion ad modification caused by evolution, and build a structure corresponding to amino acids that did not change. Finally the outside region is designed and the final structure is optimized. Such methods strictly rely on how close is the evolutionary relation among the target protein and a set of known proteins. Sequence alignments may be performed by using algorithms of global alignment, such as the Needleman and Wunsch [19], or by using local alignment algorithms such as BLAST [1]. Multiple alignments can be performed using Clustal and TCoffee algorithms [11,31].

In the next section, the main methodologies on which each of the considered tools is based are also specified.

3 Experiments on Protein Prediction

In this section we present the experimental framework we adopted to compare the performance of nine secondary structure prediction tools, that are: Jufo [18], Prof [20], Porter [21], Psipred [17], Nn-predict [14], all exploiting neural network-based approaches; HMMSTR/Rosetta [4], based on an *ab initio* method; SAM [13], based on linear hidden Markov models; Gor IV [8], using frequency analysis of amino acid conformational states; Hnn [9], based on a hierarchical and modular approach.

To begin with, we describe the data sets we have exploited in experiments and, then, we illustrate in detail the experimental tests we carried out using the aforementioned prediction tools.

3.1 Data Sets

In order to carry out a robust analysis of the performances of the analyzed tools, we considered two protein data sets.

The former data set includes 100 proteins from the *PDB25Select* database [12], a set purposely constructed to be non-redundant. Proteins in this data set are in fact characterized by a sequence identity which is less than 25%; this feature eliminates, in practice, the possible bias determined by testing homologues proteins, which could unfairly give an advantage to those techniques better working on that kind of proteins. This clearly makes the performance analysis more statistically relevant. This data set, which we will call PDB25 in the following, is shown in Figure 1(a), where proteins are identified by their PDB id [2].

The second data set we considered includes 180 proteins from the CASP 6 edition [30] and available at `http : //predictioncenter.org/`. Structures of proteins selected for CASP represent both a non-trivial and a biologically relevant test-bed for protein structure prediction tools. This data set, called CASP in the following, is reported in Figure 1(b) where, again, each protein is identified by its PDB id.

3.2 Quality Parameters

In order to test the accuracy of analyzed prediction tools, we exploited the well known SOV and $Q3$ parameters described in [23,27]. We recall that $Q3$ represents the

PDB25 Proteins									
1AAF	1ARB	1BBPA	1CAUA	1CPCL	1FHA	1GPS	1HLE	1LTSA	1PAZ
1PPFE	1TNFA	1XIMA	2BPA3	2GBIA	2MADL	2POR	3RUBS	4ICD	7APIB
9WGAA	1BBT1	1CAUB	1EGF	1FIAB	1GRCA	1HSBA	1LTSC	1MUP	1PPN
1RHD	1SNC	1YCC	2CBH	2GLSA	2MEV1	2RN2	3CD4	3SC2A	4INSB
7ZNF	2AAK	1ATX	1BBT2	1CBN	2END	1FNB	2HSDA	1LTSD	1NIPB
1PDA	1PRCC	1RND	1TABI	1TRB	256BA	2CCYA	2HAD	2MEV4	2SCPA
3CHY	3SC2B	4RCRH	8ABP	1AAPA	1AVHA	1BGC	1FXIA	1MAMH	1OIAA
1PRCM	1RPRA	1TEN	1TREA	2AAA	2CDV	2HHMA	1ABA	1CD8	1D66A
1ERP	1GSSA	1IFC	1MDAA	1RRO	1TFG	1TROA	2ACHA	2CPL	2MHU
3DFR	2SN3	3SICI	4SGBI	8ADH	1BOVA	1CDTA	1DFNA	1EZM	1GLAF

(a)

CASP Proteins									
1STZA	1SUWA	1T70A	1S12A	1SUMB	1TD6A	1T6SA	2BH8A	1VDHA	1VGGA
1WD7A	1WD5A	1WD6A	1WDEA	1WDIA	1WDJA	1WDVA	1WFXA	1WHZA	1WTYA
1WJ9A	1WJGA	1WK4A	1WKCA	1VLCA	1BNKA	1HKA	1XFKA	1WG8A	1GA7A
1G6EA	1B9KA	2A2UA	1L7CA	1VL7A	1VLOA	1J3GA	1VQUA	1XCBA	1VM0A
1VKPA	1ZWJA	2FNOA	2J85A	1E68A	1W33A	1KRHA	1EXSA	1M33A	1TR0A
1G7DA	1IZNA	1QGOA	1GGWA	1L7AA	1FW9A	1N2XA	1DW9A	1BVB	2EZM
1DPTA	1J90A	1MZHA	1KOYA	1JWEA	1G6UA	1C8UA	1TZ9A	1FZRA	1E91A
1CI8A	1BQV	1E2XA	1I8UA	1UZCA	1J83A	1QFJA	1E4FT	1Y0BA	1XQAA
1Y12A	2CV4A	1VKKA	1CCWA	1MKIA	1B7GO	1NG4A	1EUGA	1BG8A	1NO5A
1IM8A	1IZMA	1YLIA	1INO	1DBUA	1MW5A	1FWKA	1TVGA	1M3SA	1PV1A
1TE7A	1S04A	1FL9A	1MW7A	1YK3A	1N91A	1NYNA	1NKVA	1XQBA	1M2FA
1E3JA	1H7MA	1G8PA	1G291	1BL0A	1B93A	1EWQA	1Q74A	1VLPA	1NTFA
1XG8A	1XV2A	1F35A	1E54A	1RKIA	1MOPA	1GA6A	1QJVA	1YEMA	1XE1A
1BD9A	1JADA	1BYFA	1BHE	1I74A	1NBUA	1NXJA	1R9QA	1W81A	1BK7A
1C94A	1QMHA	1IJXA	1NVTA	1UE6A	1E6WA	1B0NA	1B0NB	1D3BA	1YUDA
1TZAA	1H5PA	1GAKA	1IY9A	1FC3A	1VLIA	1X9BA	1EH2	1J6UA	1UWDA
1VL4A	1O12A	2AJRA	1VLAA	1X9AA	1O0WA	1O0XA	1O0YA	2B8NA	1VKWA
1VPQA	1O13A	1U07A	1ON9A	1BKB	1GEQA	1WIWA	1WK2A	1VJVA	1QY6A

(b)

Fig. 1. The exploited protein data sets: (a) The PDB25 data set. (b) The CASP data set.

percentage of amino acids correctly predicted by a prediction tool t for a given input protein p. Conversely, the SOV parameter represents the percentage of *segments* correctly predicted by the prediction tool t for the protein p, where a segment is a portion of a secondary structure made exclusively of the same conformational state (e.g., of α-helices, or of β-strands, or of loops) and satisfies some other structural constraints. The interested reader is referred to the cited references for a more formal definition of $Q3$ and SOV.

$Q3$ and SOV measure two quite different (and often contrasting) characteristics of the predictions. Consequently, neither $Q3$ nor SOV alone are suitable to measure the overall accuracy of a tool. Thus, in order to embed into a single parameter information provided by both $Q3$ and SOV, we considered a further parameter, that we call *Prediction Accuracy*, defined as:

$$PA = \frac{2}{3} \cdot SOV + \frac{1}{3} \cdot Q3$$

which is simply obtained as a weighted mean of SOV and $Q3$; more precisely, since SOV takes into more account structural kind of information which $Q3$ does not consider, it appears sensible to weigh the SOV more than the $Q3$. Such weights have been also tested experimentally.

Tests have been carried out as follows. Each analyzed prediction tool has been required to predict the secondary structure of each considered protein. Then, each prediction has been compared with the real, known, structure to compute $Q3$, SOV and, consequently, the PA.

We next provide a description of the measures recorded in our tests; a discussion of the corresponding results is addressed in the next section.

3.3 Results

Table 1 summarizes the results we obtained by comparing different prediction tools. In particular, it shows the *average* PA scored by each prediction tool as well as the corresponding standard deviation on *(i)* the PDB25 data set, *(ii)* the CASP data set, *(iii)* all proteins included in either data sets.

Table 2 considers $Q3$ and SOV separately; in particular, it shows the average $Q3$ (resp., SOV) scored by each tool on *(i)* the PDB25 data set, *(ii)* the CASP data set, *(iii)* all proteins.

Finally, in Figure 2 some graphs are presented showing the percentage of test cases in which each predictor resulted in the set of the best ones (again, distinguishing between PDB25 and CASP data sets). The set of the best predictors results by simply considering the best performances up to a given tolerance threshold. In particular, since the differences in the PAs of the various tools are often very small we considered different levels of "tolerance" in determining the set of best predictors.

To illustrate, let p_i be an input protein, let t_j be a tool, PA_{ij} the PA of the tool t_j on protein p_i and PA_i^* the best value of PA obtained for p_i. In order to select the set of best predictors on each p_i we considered the following formula:

$$bestTools(p_i) = \{t_j \mid PA_{ij} \geq (1-\tau)PA_i^*\}$$

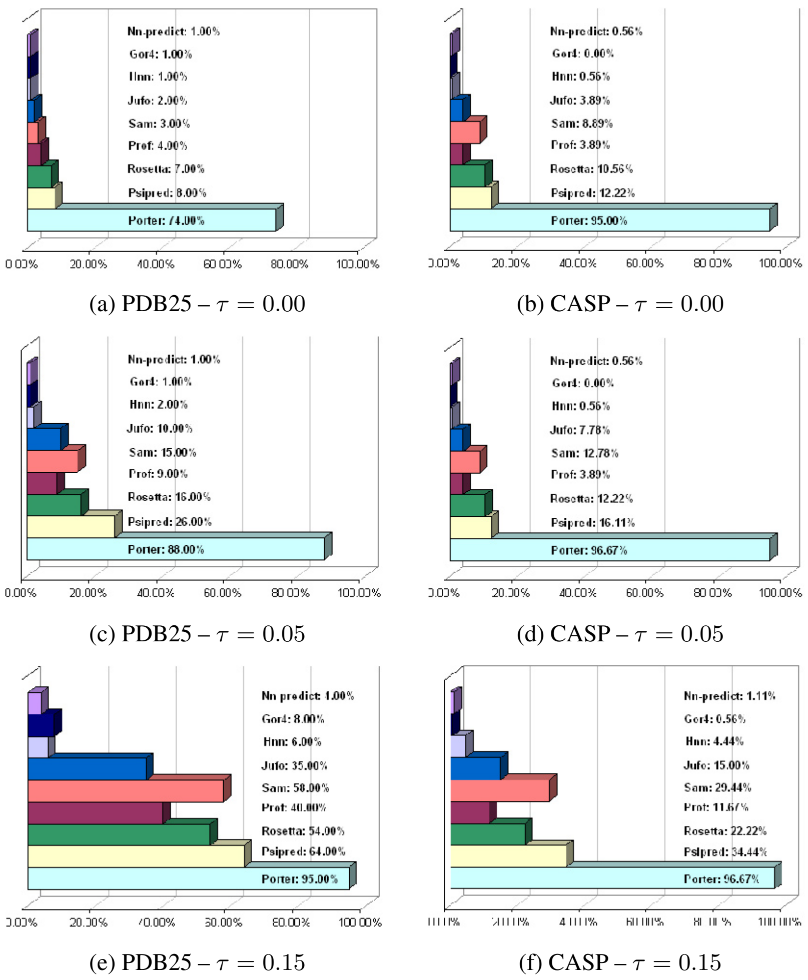
where $\tau \in [0,1]$ is the "tolerance" factor, stating the maximum tolerated deviation from the best PA. We computed the best sets of tools for each protein using the following three levels of tolerance: *(i)* $\tau = 0.00$, which selects only those predictors scoring exactly the best PA; the results for this case are shown in Figures 2(a) and 2(b). *(ii)* $\tau = 0.05$, which considers those predictors scoring a PA within 5% of the best PA; the results for this case are shown in Figures 2(c) and 2(d). *(iii)* $\tau = 0.15$, which considers those predictors scoring a PA within 15% of the best PA; the results for this case are shown in Figures 2(e) and 2(f).

3.4 Discussion

From an overall analysis of the results we obtained from experiments, we note that the PDB25 data set actually turned out to be a more demanding test-bed than the CASP set. Indeed, the average PA's measured on the CASP proteins is larger than the PA scored on PDB25 proteins for all tools except Gor IV and HMMSTR/Rosetta. This result is quite interesting because it points out that possible homologies between proteins can simplify the structure prediction process not only for those tools based on comparative modeling, but, actually, for most of them.

Table 1. Average PA and corresponding standard deviation for each prediction tool

		PDB25 Proteins		CASP Proteins		All Proteins	
Tool	**Technique**	**Avg PA**	**st.dev.**	**Avg PA**	**st.dev.**	**Avg PA**	**st.dev.**
Jufo	neural networks	70.10	13.49	74.39	10.49	72.86	11.83
Prof	neural networks	71.56	14.45	73.14	9.71	72.58	11.65
Porter	neural networks	87.04	10.69	92.10	6.85	90.29	8.77
Psipred	neural networks	77.64	11.95	80.63	9.23	79.56	10.38
Nn-predict	neural networks	55.11	11.26	57.76	10.08	56.82	10.59
HMMSTR/Rosetta	ab initio	73.43	14.07	72.84	14.01	73.05	14.03
Sam	linear hidden Markov models	75.20	13.131	79.06	9.34	77.68	11.00
Gor IV	freq. analysis of amino acid conf. states	61.21	11.96	60.81	10.11	60.95	10.81
Hnn	hierarch. and modular approach	62.46	11.74	64.25	10.68	63.61	11.10



(a) PDB25 – $\tau = 0.00$ (b) CASP – $\tau = 0.00$

(c) PDB25 – $\tau = 0.05$ (d) CASP – $\tau = 0.05$

(e) PDB25 – $\tau = 0.15$ (f) CASP – $\tau = 0.15$

Fig. 2. Percentage of times each prediction tool has been detected among the best ones at various tolerance levels

Table 2. Average $Q3$ and SOV scored by each prediction tool

	PDB25		CASP		All	
Tool	**AvgQ3**	**AvgSOV3**	**AvgQ3**	**AvgSOV3**	**AvgQ3**	**AvgSOV3**
Jufo	74.67	67.64	75.34	73.31	75.16	71.49
Prof	75.88	69.34	75.24	72.06	75.50	71.19
Porter	90.34	85.33	93.19	91.54	92.20	89.40
Psipred	81.21	76.03	82.41	79.71	82.05	78.44
Nn-predict	62.53	51.34	62.33	55.41	62.40	54.00
HMMSTR/Rosetta	79.36	70.56	75.47	71.49	76.82	71.17
Sam	79.25	73.11	80.56	78.29	80.09	76.44
Gor IV	65.87	58.93	62.83	59.77	63.92	59.43
Hnn	66.82	60.45	67.25	62.71	67.08	61.83

Second, the average performance of the analyzed tools appears notable, especially when compared to the complexity of the exploited data sets. To illustrate, while the average PA recorded for the top scoring tools in our tests agrees with the results reported in `http://cubic.bioc.columbia.edu/eva/sec/res_sec.html`, it turns out that significant improvements can still be achieved in prediction accuracy, as long as the quality attained by the tools on a protein-to-protein basis is still uncertain for almost all of them, which is well mirrored in the values of the standard deviation of the PA we recorded in our tests (see Table 1). This demonstrates that the *guarantee* for a sufficiently high precision of the prediction still remains rather low, in general, which eventually implies that the predictions yielded by those tools, even if quite good on the average, can hardly be looked at as a reliable reference by biologists.

A further interesting observation that can be drawn from Table 1 is that our results do not allow the identification of a *prominent technique* for protein secondary structure prediction, even if recent trends seem to pay much attention on neural network-based approaches. As for neural network-based techniques, in particular, on the one hand, they showed rather varying results in terms of PA (one of them, namely Porter, appears as the best predictor in the analyzed set, whereas another one of them, namely Nn-predict, scored the lowest average PA) while, on the other hand, tools based on different techniques scored competitive PAs. This, again, suggests that there is still space for improvements in protein structure prediction accuracy.

The analysis of Table 2 confirms our former claim that neither $Q3$ nor SOV alone can provide a reliable measure of the tools prediction accuracy. Indeed, the analysis of this table points out that there are cases of pairs tools (e.g., Sam and HMMSTR/Rosetta), scoring, respectively, the best $Q3$ and and the best SOV, making it difficult to figure out which one attains the best quality prediction. In this respect, our choice of defining the PA as a composed quality parameter englobing both the $Q3$ and the SOV, seems to be a sensible one.

Finally, as for the percentage of proteins upon which a predictor can be counted among the best ones, we can observe from Figure 2 that the general trend of the various tools is actually independent of the tolerance degree. Clearly, the higher the tolerance is, the higher the percentage associated with each tool will be. The interesting observation is that the *overall* ranking of the various tools is almost invariant with respect to the tolerance degree. In more detail, these graphs show that, on the employed data sets, Porter was the best performing tool both in terms of average PA and, notably, also in terms of number of successes.

4 Conclusion

In this paper we have illustrated an experimental campaign we have developed in order to compare the quality of predictions returned by protein secondary structure predictors. Analyzed predictors include the systems Jufo [18], Prof [20], Porter [21], Psipred [17], Nn-predict [14], HMMSTR/Rosetta [4], SAM [13], Gor IV [8], Hnn [9].

Two test data sets were selected: one non-redundant set (used to minimize the influence of homology amongst tested proteins on returned predictions) and a second set of proteins taken from those used in CASP, the well-known biannual protein prediction contest.

The experimental results showed that some of the available predictors are able to score quite good average accuracies, but the quest for a truly reliable automatic tool is still open, as the predictions are still characterized by significant standard deviations (which means that, for a given generic protein, the confidence for the returned prediction to have a high accuracy is largely unpredictable).

References

1. Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Reserch 25(17), 3389–3402 (1997)
2. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The Protein Data Bank. Nucleic Acids Research 28, 235–242 (2000)
3. Branden, C.I., Tooze, J.: Introduction to Protein Structure. Garland Publishing, Inc. (1998)
4. Bystroff, C., Shao, Y.: Fully automatic ab initio protein structure prediction using I-SITES, HMMSTR and HMMSTR/ROSETTA. Bioinformatics 18(suppl. 1), S54–S61 (2002)
5. Cuff, J.A., Clamp, M.E., Siddiqui, A.S., Finlay, M., Barton, G.J.: Jpred: a consensus secondary structure prediction server. Bioinformatics 14, 892–893 (1998)
6. Dudek, M., Ramnarayan, K., Ponder, J.W.: Protein structure prediction using a combination of sequence homology and global energy minimization. Journal of Computational Chemistry 2(19), 548–573 (1998)
7. Eyrich, V.A., Martin-Renom, M.A., Przybylski, D., Madhusudhan, M.S., Fiser, A., Pazos, F., Valencia, A., Sali, A., Rost, B.: Eva: Continuous automatic evaluation of protein structure prediction servers. Bioinformatics 17(12), 1242–1243 (2001)
8. Garnier, J., Gibrat, J.F., Robson, B.: GOR secondary structure prediction method version IV. Methods in Enzymology 266, 540–553 (1996)
9. Guermeur, Y.: Combinaison de classifieurs statistiques, Application a la prediction de structure secondaire des proteines., PhD Thesis (1997)
10. Guerra, C., Istrail, S.: Mathematical Methods for Protein Structure Analysis and Design: Advanced Lectures, August 13, 2003. Lecture Notes in Computer Science / Lecture Notes in Bioinformatics. Springer, Heidelberg (2003)
11. Higgins, D., Thompson, J., Gibson, T., Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressivemultiple sequence alignment through sequence weighting,position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22, 4673–4680 (1994)
12. Hobohm, U., Sander, C.: Enlarged representative set of protein structures. Protein Science 3, 522–524 (1994)

13. Hughey, R., Krogh, A.: SAM: Sequence alignment and modeling software system., Technical Report UCSC-CRL-95-7, University of California, Santa Cruz, CA (1995)
14. Kneller, D.G., Cohen, F.E., Langridge, R.: Improvements in protein secondary structure prediction by an enhanced neural network. Journal of Molecular Biology 214, 171–182 (1990)
15. Lesk, A.M.: Introduction to Protein Architecture. Oxford University Press (2001)
16. Lin, K., Simossis, V.A., Taylor, W.R., Heringa, J.: A Simple and Fast Secondary Structure Prediction Algorithm using Hidden Neural Networks. Bioinformatics 21(2), 152–159 (2005)
17. McGuffin, L.J., Bryson, K., Jones, D.T.: The PSIPRED protein structure prediction server. Bioinformatics 16, 404–405 (2000)
18. Meiler, J., Mueller, M., Zeidler, A., Schmaeschke, F.: JUFO: Secondary Structure Prediction for Proteins (2002), http://www.jens-meiler.de
19. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 442–453 (1970)
20. Ouali, M., King, R.D.: Cascaded multiple classifiers for secondary structure prediction. Protein Science 9, 1162–1176 (2000)
21. Pollastri, G., McLysaght, A.: Porter: a new, accurate server for protein secondary structure prediction. Bioinformatics 21(8), 1719–1720 (2005)
22. Rost, B., Sander, C.: Prediction of protein secondary structure at better than 70% accuracy. Journal of Molecular Biology 232, 584–599 (1993)
23. Rost, B., Sander, C., Schneider, R.: Redefining the goals of protein secondary structure prediction. Journal of Molecular Biology 235, 13–26 (1994)
24. Simons, K.T., Bonneau, R., Ruczinski, I., Baker, D.: Ab Inition Protein Structure Prediction of CASP III targets using Rosetta. Proteins (suppl. 3), 171–176 (1999)
25. Tramontano, A.: The Ten Most Wanted Solutions in Protein Bioinformatics. CRC Press, Boca Raton (2005)
26. Tramontano, A., Lesk, A.M.: Protein Structure Prediction: Concepts and Applications. John Wiley & Sons, Chichester (2006)
27. Venclovas, C., Zemla, A., Fidelis, K., Moult, J.: Some measures of comparative performance in the three CASPs. Proteins: Structure, Function, and Genetics 34, 220–223 (1999)
28. Webster, D.M.: Protein Structure Prediction: Methods and Protocols (Methods in Molecular Biology). Humana Press, August 15 (2000)
29. White, J.V., Stultz, C.M., Smith, T.F.: Protein Classification by Stochastic Modeling and Optimal Filtering of Amino-Acid Sequences. Mathematical Biosciences 119, 35–75 (1994)
30. Critical assessment of techniques for protein structure prediction, http://predictioncenter.llnl.gov/casp6/Casp6.html
31. TCoffee Multiple Alignments tools. Swiss institute of bioinformatics, http://www.tcoffee.org/

Workshop on Using Emerging Parallel Architectures for Computational Science

Workshop on Using Emerging Parallel Architectures for Computational Science

Bertil Schmidt and Douglas Maskell

School of Computer Engineering, Nanyang Technological University, Singapore 639798
{asbschmidt,asdouglas}@ntu.edu.sg

Abstract. The Workshop on *Using Emerging Parallel Architectures for Computational Science*, held in conjunction with ICCS 2009, provides a forum for exploring the capabilities of emerging parallel architectures such as GPUs, FPGAs, Cell B.E., and multi-cores to accelerate computational science applications.

Keywords: Computational Science, Parallel Computer Architectures, GPGPU, Reconfigurable Computing, Heterogeneous Multi-cores, High Performance Computing.

1 Introduction to the Workshop

Welcome to the Workshop on *Using Emerging Parallel Architectures for Computational Science*. This workshop has been motivated by the significant transformation of the computing landscape in recent years with the emergence of more powerful processing elements such as GPUs, FPGAs, Cell B.E., multi-cores, etc. On the multi-core front, Moore's Law has transcended beyond the single processor boundary with the prediction that the number of cores will double every 18 months. Going forward, the primary method of gaining processor performance will be through parallelism. Multi-core technology has visibly penetrated the global market. Accordingly to the Top500 lists, the HPC landscape has evolved from supercomputer systems into large clusters of dual or quad-core processors. Furthermore, GPUs, FPGAs and heterogeneous multi-cores have been shown to be formidable computing alternatives, where certain classes of applications witness more than one order of magnitude improvement over their GPP counterpart. Therefore, future computational science centers will employ resources such as FPGAs, GPUs and Cell architectures to serve as co-processors to offload appropriate compute intensive portions of applications from the servers. This workshop provides a forum for exploring the capabilities of emerging parallel architectures to accelerate computational science applications.

The technical program was put together by the Workshop Chairs Bertil Schmidt and Douglas Maskell and 19 members of a distinguished program committee. The workshop received 23 submissions. After an initial screening 21 submissions were reviewed by at least three experts in the field. Based on the reviews, 16 papers were selected for presentation at the workshop and inclusion in the workshop proceedings.

We wish to thank the program committee members for submitting thoughtful reviews and all authors who submitted high-quality manuscripts. We plan to continue the workshop next year.

2 Workshop Organizers

Workshop Co-Chairs:

- Bertil Schmidt (Nanyang Technological University, Singapore)
- Douglas Maskell (Nanyang Technological University, Singapore)

Program Committee:

- Michael Huebner (University of Karlsruhe, Germany)
- Manfred Schimmler (University of Kiel, Germany)
- David Luebke (NVIDIA, USA)
- Simon See (SUN Microsystems)
- Neil Bergmann (University of Queensland, Australia)
- Philip Leong (Chinese University of Hong Kong, Hong Kong)
- Heiko Schroder (RMIT University, Australia)
- Alexandros Stamatakis (TU Munich, Germany)
- Dominique Lavenier (IRISA, France)
- Tarek El-Ghazawi (George Washington University, USA)
- Jaroslaw Zola (Iowa State University, USA)
- Michela Taufer (University of Delaware, USA)
- Rick Goh (IHPC, Singapore)
- Scott Emrich (University of Notre Dame, USA)
- Ananth Kalyanaraman (Washington State University, USA)
- Shi Haixiang (NTU, Singapore)
- Gerrit Voss (Fraunhofer IGD, Germany and NTU, Singapore)
- Weiguo Liu (NTU, Singapore)
- Malcolm Low (NTU, Singapore)

3 List of Accepted Papers

1. *Solving Sparse Linear Systems on NVIDIA Tesla GPUs*, M. Wang, H. Klie, M. Parashar, H. Sudan
2. *Multi-walk Parallel Pattern Search Approach on a GPU Computing Platform*, W. Zhu, J. Curry
3. *Pairwise Distance Matrix Computation for Multiple Sequence Alignment on the Cell Broadband Engine*, A. Wirawan, B. Schmidt, C.K. Kwoh
4. *Evaluating the Jaccard-Tanimoto Index on Multi-Core Architectures*, V. Sachdeva, D.M. Freimuth, C. Mueller
5. *A particle-mesh integrator for galactic dynamics powered by GPGPUs*, D. Aubert, M. Amini, R. David
6. *Evaluation of the SUN UltraSparc T2+ Processor for Computational Science*, M. Sandrieser, S. Pllana, S. Benkner

7. *Streamlining Offload Computing to High Performance Architectures*, M. Purcell, O. Callanan, D. Gregg
8. *Power Consumption of GPUs from a Software Perspective*, C. Collange, D. Defour, A. Tisseran
9. *A Note on Auto-tuning GEMM for GPUs*, Y. Li, J. Dongarra, S. Tomov
10. *Parallel calculating of the goal function in metaheuristics using GPU*, W. Bozejko, C. Smutnicki, M. Uchronski
11. *Fast Conjugate Gradients with Multiple GPUs*, A. Cevahir, A. Nukada, S. Matsuoka
12. *CUDA solutions for the SSSP problem*, P.J. Martín, R. Torres, A. Gavilanes
13. *Experiences With Mapping Non-Linear Memory Access Patterns into GPUs*, E. Gutierrez, S. Romero, M.A. Trenas, O. Plata
14. *A Massively Parallel Architecture for, Bioinformatics*, G. Pfeiffer, S. Baumgart, J. Schroder, M. Schimmler
15. *Accelerated Discovery of Discrete M-Clusters/Outliers on the Raster Plane using Graphical Processing Units*, C. Trefftz, J. Szakas, I. Majdandzic, G. Wolffe
16. *GPU accelerated RNA folding algorithm*, G. Rizk, D. Lavenier

Solving Sparse Linear Systems on NVIDIA Tesla GPUs*

Mingliang Wang[1], Hector Klie[2], Manish Parashar[1], and Hari Sudan[2]

[1] NSF Center for Autonomic Computing (CAC)
The Applied Software System Laboratory (TASSL)
Rutgers, The State University of New Jersey
94 Brett Road, Piscataway, NJ 08854, USA
{mlwang,parashar}@cac.rutgers.edu
[2] ConocoPhilips
Houston, TX, USA
{Hector.M.Klie,Hari.H.Sudan}@conocophillips.com

Abstract. Current many-core GPUs have enormous processing power, and unlocking this power for general-purpose computing is very attractive due to their low cost and efficient power utilization. However, the fine-grained parallelism and the stream-programming model supported by these GPUs require a paradigm shift, especially for algorithm designers. In this paper we present the design of a GPU-based sparse linear solver using the Generalized Minimum RESidual (GMRES) algorithm in the CUDA programming environment. Our implementation achieved a speedup of over 20x on the Tesla T10P based GTX280 GPU card for benchmarks with from a few thousands to a few millions unknowns.

1 Introduction

Single core performance is leveling off, and recent growth has focused on on-chip parallelism and multiple cores. Chip designers are increasing processing capability by building multiple processing cores in one chip package and keep increasing the numbers of cores that can fit into a chip. Moore's Law has a new interpretation: it is the number of cores that doubles every 18 months [1]. Following this trend, commodity hardware is delivering formidable processing power at rather moderate cost and GPU is taking the lead. For example, the latest T10P GPU core embedded in a NVIDIA GTX280 graphics card delivers over a TFLOP [2], at a price tag of less than $500. Other competing venders in graphics and general purpose computing have similar offerings.

* The research presented in this paper was supported in part by National Science Foundation via grants numbers IIP 0758566, CCF-0833039, DMS-0835436, CNS 0426354, IIS 0430826, and CNS 0723594, and by Department of Energy via the grant number DE-FG02-06ER54857, and was partially conducted as part of the NSF Center for Autonomic Computing at Rutgers University. The authors would also like to thank NVIDIA for their technical support.

The huge performance available on GPUs has generated interest in using it for general purpose computing [3,4,5]. GPUs are stream processors and were historically designed for shader operations that involve applying a series of shader kernels to a large collection of data elements called a stream. This is a restricted form of parallelism, and while it simplifies the hardware and software required to implement it, mapping general-purpose computing to this paradigm is often difficult.

In this paper, we explore the use of GPUs for solving large sparse linear systems using the Generalized Minimal RESidual (GMRES) [6] algorithm. This is an important algorithm because many scientific applications can be transformed into solving a sparse linear system, and GMRES is the classical iterative method when the coefficient matrix is asymmetric. Our preconditioned GMRES implementation achieves a speedup of over 20x on the Tesla T10P based GTX280 GPU card, relative to a serial version[1], for benchmarks with from a few thousands to a few millions unknowns. Close to the highest speedup was achieved for the largest input.

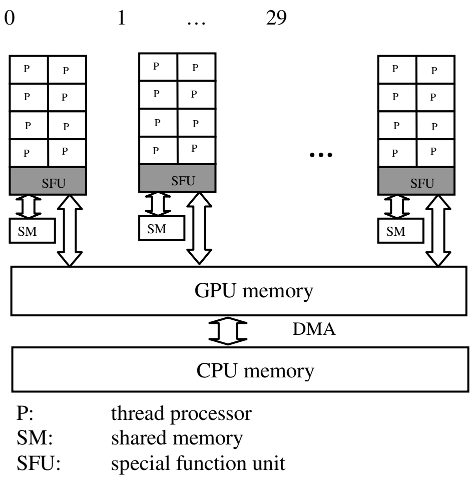


Fig. 1. Architecture overview of a Tesla T10P core for general-purpose computation

2 Backgrounds and Related Work

An architectural overview of the TESLA T10P GPU architecture, from the perspective of general purpose computing, is presented in Fig. 1. Note that graphics related details are omitted.

Tesla T10P SIMD core is an SIMD array of scalar processors. It has 30 multiprocessors with 8 Thread Processor (cores) each, resulting in a total of 240

[1] The serial version of GMRES was based on the FORTAN version in SPARSEKIT, which was manually translated to ANSI C. The C version performed the same or better than the FORTRAN version for all benchmark inputs. The code was compiled using the Intel® *icc* version 10.1 compiler with optimization flag –O2, and executed on the Intel® Harpertown core clocked at 3.0GHz. Optimization flag –O2 was chosen because the codes performed slightly better using –O2 as compared to using –O3.

cores. Each multiprocessor has one instruction issue unit, which broadcasts instructions to the scalar cores [2,7]. The flexibility offered by T10P cores includes the ability to execute scalar threads with arbitrary branch behaviors and memory access patterns. In [2], this exposure of the SIMD units as individual programmable scalar units is called Single Thread Multiple Thread (SIMT). However this flexibility comes at a cost if it is not exploited properly [2,7].

The GPU platform also has a more complex memory structure. In case of the T10P, cores have access to a set of off-chip memory spaces: global memory, local memory, constant memory, and texture memory. Constant and texture memory are cached but global and local memory are not. The on-chip memory includes a small shared memory (16KB on a T10P chip) that is local to each multiprocessor and can be used as software managed cache. The instruction cache is transparent to programmers. For general purpose computing on a GPU, most programmers only use global memory, shared memory, and local memory. The latency of a global memory access is about 400 core clock cycles, and that of a shared memory access is 1 cycle if there are no bank conflicts [7].

The connection between CPU memory and GPU memory is through a fast PCIe 16x point-to-point link. The transfer time for a given size data chunk can be computed as the following.

$$t_{transfer} = t_c + \frac{S}{B} \tag{1}$$

t_c : the constant startup overhead. Device to host: $9.95\mu s$, host to device: $13.3\mu s$

S : bytes transfered

B : bandwidth. Device to host: 5.76GB/s, host to device: 5.05 GB/s

Note that the above equations as well as the constants are derived using actual transfer times measured for a range of data sizes (1024 B to 64 MB).

2.1 Support for Programming on GPU

Programming GPUs for general purpose computing is not easy. Earlier programming models were based on shading languages such as HLSL, GLSL [8] and Cg [9], which are graphics oriented and have to be used with the OpenGL or DirectX APIs. In these models, computation was essentially organized as a sequence of *shading operations* (kernels) on graphics data (streams). Programmers had to make explicit calls to these graphics APIs to manage streams and launch kernel calls.

With the advent of Brook [10], direct general-purpose stream programming is supported on the GPU platform. General-purpose computation on a GPU is directly expressed as *kernels* acting on *streams* using this model. The model also eliminates the unnatural graphics API calls in the prior models. CUDA [7] added even more flexibility by allowing scattering, which enables direct and random access to DRAM associated with a GPU and makes programming on GPU almost as flexible as a CPU. Other features of CUDA include the exposure of the on-chip fast shared memory, local synchronization, and atomic operations on memory.

2.2 Related Work

In the area of matrix computation, Galoppo et al. [11] studied the peak performance of a GPU dense matrix decomposition algorithm. Volkov et al. [12] also investigated the implementation of dense matrix computation and claimed to approach peak performance on the G80 series GPUs using their matrix-matrix multiply routine. In [13], Garland gave an introduction to sparse matrix algorithms on NVIDIA GPUs. Data parallel approach to GPU programming using the primitive segmented scan was presented in [14]. Boltz el al. [15] discussed an iterative solver for symmetric sparse matrix. These previous efforts focused on handling the peculiarities of the GPU platform while implementing a particular algorithm or algorithmic building block using the programming environment provided by the vendors. They showed impressive results and demonstrated the tremendous potential of GPUs as coprocessors.

GMRES is a widely used iterative method for solving large linear systems, and its parallelization has been attempted on various platforms using different approaches. In [16], a hybrid parallelized GMRES on GRID systems was presented. Compared to our method, it is a distributed memory implementation and involves computation of eigenvalues that requires more computation and does not efficiently use the GPU hardware. An earlier parallel implementation using PVM message passing was reported in [17]. It was programmed using a parallel version of basic linear algebra operations for distributed memory computers. It was designed for the linear systems derived from 5-point finite difference discretizations and exploited its particular communication patterns for efficiency. Our implementation nonetheless is for generic sparse linear systems.

Conjugate gradient is a related iterative algorithm that solves symmetric positive-definite linear systems, and its implementation without preconditioning on an earlier NVIDIA GPU platform was reported in [15]. This implementation was based on an awkward graphics API and focused on issues that arose while mapping the algorithms to this unwieldy API. Our preconditioned GMRES solves generic sparse matrixes and has a powerful parallelized preconditioner.

3 Parallelizing GMRES for Stream Processing Using GPUs

A key aspect of programming applications to use GPUs involves partitioning the computation into a partition that runs on the CPU and another partition that runs on the GPU. The purpose of this partitioning is to offload selected, for example, compute intensive, portions to the GPU, while still using the CPU for other aspects such as global synchronization. Note that typically it is not beneficial to offload a complete application to the GPU due to the limitations of the stream-processing model. Stream processing is most appropriate when similar operations (kernels) are applied to a large collection of homogeneous data elements. A kernel is applied independently to each element without communication during execution. This limits the computation that can be implemented as a single kernel and therefore, most applications require a sequence of kernel calls. Additionally, an application may need to communicate with other entities or use other services, which may not be accessible from a GPU.

Application partitioning presents several challenges. It requires programmers to manually identify which computations are appropriate for the GPU and to keep track of the computations running on the CPU and on the GPU, as well as manually initiate and manage data transfers between the two. This style of programming is not only tedious but also error-prone, and significantly impacts programmers' productivity.

To address this issue, we propose to create a high-level library that can transparently offload compute-intensive operations to GPU for certain classes of applications, and that presents, on the CPU side, a set of abstract data types as basic building blocks for programming numeric applications. These abstractions include *vectors*, *dense matrices* and *sparse matrices*. These data types maintain memory regions within GPU and CPU memory spaces and support synchronization between them. A set of basic operations similar to the BLAS level 1, are included. Problem specific operations such as those used in GMRES are implemented over these basic operations. The method to parallelize GMRES for the Tesla T10P GPU using CUDA is presented below.

3.1 GMRES Iterations

GMRES is a well-known iterative method to solve large sparse non-symmetric systems of the form $Ax = b$ and was proposed by Saad and Schultz in 1986 [16]. GMRES generates a basis of dimension m for the Krylov subspace $K_m(A,v) = span\{v, Av, A^2V, ..., A^{m-1}v\}$ for a given initial residual v. The Krylov basis is constructed with the Arnoldi algorithm which is typically implemented in terms of the modified Gram-Schmidt orthogonalization process to generate the orthogonal basis $\{v_1, v_2, ..., v_m\}$ of the Krylov subspace $K_m(A,v)$.

Given the Krylov basis, an intermediate solution x_m in the m-*th* iteration is given by the solution of the least squares problem: minimize $\|b - Ax\|_2$, where $x \in x_0 + K_m(A, r_0)$ and $r_0 = b - Ax_0$ is the residual associated with the initial guess x_0.

As m increases, the computational cost increases at least as $O(m^2n)$, and memory cost increases as $O(mn)$. In large systems, this limits the largest value of m that can be used. Restarted GMRES is applied to remedy this situation. After a fixed number m of iterations, the solution x_m is used as the initial guess x_0 to restart a new GMRES.

However, GMRES does not always converge, and even when it converges, it may take too many iterations to reach the desired residual tolerance. Preconditioning is a technique to improve this situation. It replaces the system $Ax = b$ with the modified systems

$$M^{-1}Ax = M^{-1}b. \tag{2}$$

or

$$AM^{-1}\hat{x} = b,\ x = M^{-1}\hat{x}\ . \tag{3}$$

The desired properties of M include 1) the modified system $M^{-1}A$ converges fast, 2) linear systems $My = c$ easy to solve; 3) easy to parallelize. The preconditioned GMRES algorithm is shown in **List 1**.

The most expensive operations of this algorithm are computing the term $w = AM^{-1}v_j$. It requires solving a linear system with coefficient matrix M and a matrix-vector multiplication. In parallelizing this algorithm, we choose to find a preconditioner that makes computing the vector term $w = AM^{-1}v_j$ amenable to be distributed in different processors. For the time being, we do not seek parallelizing the GMRES algorithm itself.

List 1. Left-preconditioned Generalized Minimum Residual algorithm with restarts

```
START:
compute $r_0 = b - AM^{-1}x_0$,
$\beta = \|r_0\|$, and $v_1 = r_0 / \beta$
for j=0,1,2,…m do
    compute $w = AM^{-1}v_j$
    for i=1,2,3,…,j do
        $h_{ij} = v_i^T w$
        $w = w - h_{ij}w_i$
    end do
    compute $h_{j+1,j} = \|w\|$ and $v_{j+1} = w / h_{j+1,j}$
end do
//solve the least square problem $y_m$ such that it is
$\min_y \|\beta e_1 - \hat{H}y\|$, $\hat{H}$ is the matrix constructed from $h_{ij}$//
$x_m = x_0 + V_m y_m$
//test for termination condition //
if satisfied,
    goto DONE
else
      $x_0 = x_m$
      goto START
end if
DONE:
output $x = M^{-1}x_m$
```

3.2 Parallelizing Matrix Vector Multiplication and Pre-conditioning

Sparse matrix vector multiplication appears in computing the term $w = AM^{-1}v_j$, and hence requires us to choose a strategy that is compatible with the parallel solve operation in the preconditioning matrix.

In the serial GMRES, incomplete LU decomposition is often used as the preconditioner as it can greatly improve the convergence rate. It is illustrated in the following equation.

$$A = LU + E \,. \tag{4}$$

For a sparse matrix A, it is LU-decomposed but only at the locations where A originally has non-zeros. The term E accounts for the dropped elements.

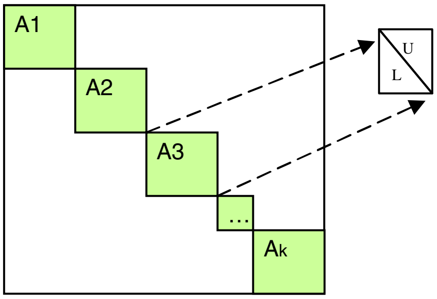


Fig. 2. Block ILU preconditioner is less effective than the conventional ILU, but has plenty of parallelism to exploit. This fits well in the data parallel model of computation on GPU. Each individual block is sparse and back/forward substitutions are used to compute $U^{-1}L^{-1}v$. The typical block size used in the experiments was 32.

We parallelize ILU by using its variant block-ILU [19], which incurs further drops by conducting ILU only along the main diagonal. Coefficient matrix A is divided into equal sized sub-matrices and then locally decomposed using ILU, as shown in Fig. 2.

Because these blocks do not communicate to each others in decomposition and also in solving it, this scheme fits well in the data parallelism paradigm. A stream now is a collection of sub-matrices along the main diagonal.

With this parallelizing scheme for the preconditioner, we can now easily compute in parallel the sparse matrix vector multiplication by computing a lot of small matrix vector multiplication on the main diagonal blocks. General sparse matrix vector multiplication is investigated in [13].

However, the preconditioner computed from this block ILU is in general less effective than the conventional ILU computed from the whole matrix and hence requires more iterations to reach the same level of residual.

3.3 Polynomial Preconditioner

A preconditioner can be enhanced by computing a polynomial of it and use this new matrix as the preconditioner. This technique is discussed in [19]. The following is a polynomial preconditioner of order *s*.

$$M^{-1} = [I + N + ... + N^s]\,. \tag{5}$$

with

$$N = [I - \omega U^{-1} L^{-1} A]\,. \tag{6}$$

We implement a polynomial preconditioner using the block ILU preconditioner and empirically compare its effectiveness.

Note that this polynomial preconditioner is not computed explicitly but through a series of forward-backward substitutions and matrix vector multiplications, built from the operations discussed in above.

4 Experimental Evaluation

The parallel GMRES was tested on the Tesla T10P GPU using a set of matrix data from the oil field simulation data of ConocoPhillips. The order of the system ranges from ~2000 to ~1.1 million. The serial version was run on Intel Harpertown clocked at 3.0GHZ and the parallel version on the same machine, with a NVIDIA GTX 280 GPU card. They were compiled using the Intel ICC version 10.1 and used CUDA SDK version 2.0. Table 1 presents the experimental results for different matrices. Residual tolerance 1e-2 reflects the actual value requested in a modeling application at ConocoPhillips.

Table 1. Experimental evaluation of GMRES on Tesla T10P GPU. Polynomial order=1, GMRES restart=20, final residual=1e-2. Times are in seconds.

Matrix	N	Serial Iters.	Parallel Iters.	Serial Time	Parallel Time	Speedup	Mem Trans.	Diff. bewteen Solutions [2]
Sample 1	1946	3	6	0.0018	0.0060	0.3	5.30%	1.8e-05
Sample 2	192096	10	20	9.41	1.16	8.1	2.30%	8.0e-03
Sample 3	184102	13	19	7.67	0.326	23.5	3.60%	1.3e-04
SPE 10_1	132000	40	40	4.82	0.70	6.8	2.70%	2.4e-02
SPE 10_2	1122000	140	100	436.33	22.10	19.7	1.30%	2.1e-03

The effect of polynomial enhancement is shown in Fig. 3, for one of the matrices. As evident in the figure, polynomial enhancement is effective, i.e., as the order of the polynomial increases, the number of iterations required to reach a desired precision decreases. However, as the order increased, each iteration requires more computation. Our empirical study showed that a low order, i.e., 1~2, yields the best result for the all the matrices tested.

[2] Difference of solutions is measured as ||x1 – x2||/||x2|| (2-norm), where x1 and x2 are the parallel and serial solutions respectively.

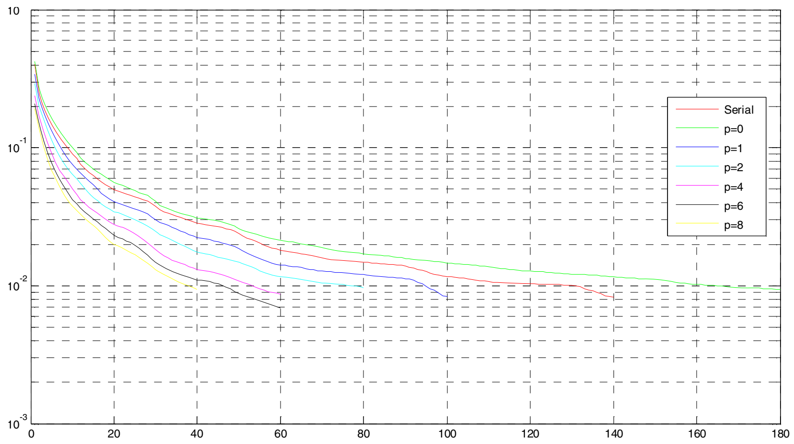


Fig. 3. Effectiveness of polynomial enhancement. Matrix=85x220x60, n=1,122,000, number of non-zero = 7,780,000, restart=20.

5 Conclusions

The parallel implementation of GMRES on the Tesla T10P GPU presented in this paper has shown very encouraging speedups (up to ~ 20x), which confirms the great potential benefits of offloading computation to GPUs. In parallelizing GMRES, block-ILU plays a crucial role, as it not only parallelizes the preconditioning computation but also naturally decomposes the matrix-vector multiplication and the subsequent polynomial enhancement. Discovering this strategy requires deep understanding of the iterative algorithms and also the characteristics of the GPU hardware. This highlights the necessity for collaborations between domain experts and parallel program developers, as well as the need for higher-level programming support.

In the empirical study presented, it was observed that a low degree ($s < =2$) was a well-balanced choice from the perspective of effectiveness of polynomial enhancement. When the degree is larger, even though the preconditioner is still improves, the additional computation incurred dominates the gains due to reduced iterations.

In our implementation using C++, the serial version of GMRES was around 1000 lines of codes, while the parallel one was roughly 5000 lines of code. From the productivity standpoint, this partly illustrates the additional work needed in parallel programming on GPU with CUDA.

The small C++ library developed helped hide the details of GPU programming for numerical algorithms from the applications experts and enabled the access of the GPU platform from a broader audience.

References

1. Agarwal, A., Levy, M.: The kill rule for multicore. In: Proceedings of the 44th annual conference on Design automation, San Diego, California, pp. 750–753. ACM, New York (2007)
2. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A unified graphics and computing architecture. Micro, IEEE 28(2), 39–55 (2008)
3. He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N., Luo, Q., Sander, P.: Relational joins on graphics processors, Vancouver, Canada, pp. 511–524. ACM, New York (2008)
4. He, B., Govindaraju, N.K., Luo, Q., Smith, B.: Efficient gather and scatter operations on graphics processors. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, Reno, Nevada, pp. 1–12. ACM Press, New York (2007)
5. Rodrigues, C.I., Hardy, D.J., Stone, J.E., Schulten, K., Hwu, W.M.W.: GPU acceleration of cutoff pair potentials for molecular modeling applications. In: Proceedings of the 2008 conference on Computing frontiers, Ischia, Italy, pp. 273–282. ACM, New York (2008)
6. Saad, Y., Schultz, M.H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on Scientific and Statistical Computing 7(3), 856–869 (1986)
7. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA, Los Angeles, California, pp. 1–14. ACM Press, New York (2008)
8. Rost, R.J.: OpenGL(R) Shading Language. Addison-Wesley Professional, Reading (2004)
9. Mark, W.R., Glanville, R.S., Akeley, K., Kilgard, M.J.: Cg: a system for programming graphics hardware in a c-like language. In: ACM SIGGRAPH 2003 Papers, San Diego, California, pp. 896–907. ACM Press, New York (2003)
10. Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for GPUs: stream computing on graphics hardware. ACM Trans. Graph. 23(3), 777–786 (2004)
11. Galoppo, N., Govindaraju, N.K., Henson, M., Manocha, D.: LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing., p. 3. IEEE Computer Society, Los Alamitos (2005)
12. Volkov, V., Demmel, J.: LU, QR and Cholesky factorizations using vector capabilities of GPUs. Technical Report UCB/EECS-2008-49, EECS Department, University of California, Berkeley (May 2008)
13. Garland, M.: Sparse matrix computations on manycore GPU's, Anaheim, California, pp. 2–6. ACM, New York (2008)
14. Sengupta, S., Harris, M., Zhang, Y., Owens, J.D.: Scan primitives for GPU computing. In: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, San Diego, California, pp. 97–106. Eurographics Association (2007)
15. Bolz, J., Farmer, I., Grinspun, E., Schroder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In: ACM SIGGRAPH 2003 Papers, San Diego, California, pp. 917–924. ACM, New York (2003)
16. Zhang, Y., Bergere, G., Petiton, S.: A parallel hybrid method of GMES on grid system. In: Parallel and Distributed Processing Symposium, IPDPS 2007, IEEE International, pp. 1–7 (2007)
17. Dias, R., Cunha, D., Hopkins, T.: A parallel implementation of the restarted GMRES iterative algorithm for nonsymmetric systems of linear equations. Adv. Comp. Math 2, 261–277 (1994)
18. He, H., Bergere, G., Petiton, S.: GMRES method on lightweight grid system. In: Proceedings of the 4th International Symposium on Parallel and Distributed Computing, pp. 74–82. IEEE Computer Society Press, Los Alamitos (2005)
19. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics (April 2003)

A Particle-Mesh Integrator for Galactic Dynamics Powered by GPGPUs

Dominique Aubert[1], Mehdi Amini[2], and Romaric David[2]

[1] Observatoire Astronomique, Universite de Strasbourg, France
[2] Direction Informatique, Universite de Strasbourg, France

Abstract. We present a particle-mesh N-body integrator running on GPU using CUDA. Relying on a grid-based description of the gravitational potential, it can simulate the evolution of self-interacting 'stars' in order to model e.g. galaxies. All the steps of the application have been ported on the GPU , namely 1/ an histogramming algorithm with CUDPP, 2/ of the resolution of the Poisson equation by means of FFT with CUFFT and multi-grid relaxation, 3/ of an optimized finite-difference scheme to compute the accelerations of stars and 4/ of an update procedure for positions and velocities. We present several tests at different resolution, and reach a speedup from 2 to 50 depending on the resolution and on the test case.

1 Introduction

By essence, astrophysics lack of laboratory experiments and from this intrinsic limitation emerges the need to rely on numerical simulation in order to understand the observations. Among the different fields of astrophysics, galactic dynamics has been a playground for numerical simulations for almost 50 years and it has been accompanied by numerical cosmology which ignited some of the largest scientific calculations ever made (see e.g. [1,2,3]). They both heavily relies on the use of N-Body integration techniques. Among the latter, one can cite direct N-Body integration (PP hereafter), Particle-Mesh (PM) and its extensions (P3M, AP3M), Tree-codes and AMR integrators. The recent introduction of ready-to-use API for General Purpose Graphical Processor Units (GPUs hereafter) will strongly impact this domain by providing an easy way to boost the performances of existing codes. Numerical experiments might be made faster and incidentally larger and hopefully more realistic. Several implementations of PP-methods (see e.g. [4,5]) have previously been made available on GPU. In such integrators, pairwise interactions between stars are explicitly computed, implying a $O(n^2)$ complexity and thus limiting the number n of particles taken into account (typically $10^4 - 10^5$ on a single machine). We present here an attempt to fully develop on GPU a Particle-Mesh integrator for galactic dynamics, a method that can easily model the evolution of millions of bodies. Using CUDA, the API developped by NVidia for its devices, the overall speedup with respect to pure CPU computation spans from 2 to 50 thus promising interesting perspectives for future simulations. First, the principles of PM integrators and the parallelization are briefly described in section II. Performances compared to CPU computation are presented in the following section, before the conclusion.

2 An Overview of the Particle-Mesh Integrator

The purpose of N-Body integrations is to simulate the mechanical evolution of a dynamical system due to its inner (an sometimes external) interactions. The motion of a 'star'x at position **x** and velocity **v** is modified as time goes by according to the laws of motion:

$$\mathbf{x}(t+dt) = \mathbf{x}(t) + \mathbf{v} \times dt \quad (1)$$

$$\mathbf{v}(t+dt) = \mathbf{v}(t) + \gamma \times dt \quad (2)$$

$$\gamma = -\nabla\phi, \quad (3)$$

where γ stands for the star's acceleration and $\phi(\mathbf{x})$ stands for the gravitational *potential* applied to the star at its location. The potential is a scalar field and is related to the spatial distribution of matter via the Poisson equation $\nabla^2\phi = 4\pi G\rho(\mathbf{x})$, where $\rho(\mathbf{x})$ stands for the density of stars at a position **x**. The density can be computed from the knowledge of the stars positions, which in turn makes it possible to predict the motion of these bodies through the evaluation of the potential. A handful of methods exist to solve this situation (see e.g. [6,7] for a review in an astronomical context) and the following sections will focus on the so-called particle-mesh method (PM hereafter).

Running a PM integrator with realistic problem sizes (128^3 and larger) on the researchers desktop machine can be time consuming. We aim at setting the foundations of a fast PM integrator able to run large simulations on desktop machines. It would ease the access to the results of large and more accurate simulations and/or accelerates the mass production of simulated catalogs by

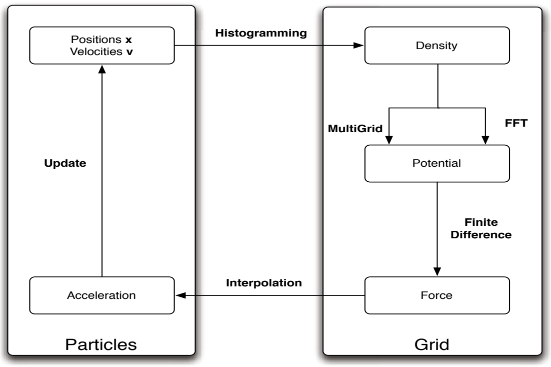


Fig. 1. Flow of operations in a PM calculation. One cycle corresponds to one time step. Operations on the left hand side of the diagram act on 'particles' data, while the right hand side operations act on fixed grid data.

quickly providing large sets of small numerical experiments. We implemented all of the steps involved in such calculation on GPU with CUDA, as the SIMD programming mode and the libraries of the CUDA toolkits could fit the steps of our method. We mostly take advantage of the multithreading abilities of these devices to accelerate the computation.

PM-driven N-Body integrations loop over a well-defined sequence of elementary steps, which can be listed as follow and summarized in figure 1 (see e.g. [6,7,8] for details):

1. Density evaluation : knowing the position of the particles, the density ρ is evaluated on a 3D regular grid. Following an algorithm described by [8], we implemented an SIMD Nearest-Grid-Point assignment scheme. It relies on the radix sort and the scan primitives included in the CUDPP library. We also used a 'mixed' algorithm which still uses the GPU for finding the nearest cells to the particles but updates the density on the CPU. It simplifies the calculations but involves recurrent data transfers between the host and the device.
2. Potential evaluation: the density ρ being available, the potential is computed on the same 3D grid via the resolution of the Poisson equation. It is usually achieved with FFT [8] or multi-grid (MG hereafter) relaxation [9]. MG techniques are less efficient than FFT but can be used for any kind of boundary conditions (while FFT assumes periodic computational domain). MG can also solve the modified gravity versions of the Poisson Equation [10]. Parallelizations of both have been widely studied and implemented [11,12]. We developed both versions on the GPU, using the CUFFT API for FFT and writing from scratch a MG solver.
3. Accelerations calculation: the forces in the 3D grid are directly available from the potential using finite differentiation. We adapted this entire step to the GPU and optimized it from a SIMD point-of-view while taking into account data locality problems.
4. Interpolation: the data representation switches back to a particle description. Each body is being independently assigned an acceleration by interpolation at its position of the 3D grid of forces.
5. Velocities and positions update: the accelerations lead to the update of the velocities and the velocity update allows to update the positions. In practice, we used a common leapfrog scheme, where velocities and positions are updated in a staggered fashion. This step is parallel by nature as each particle is being assigned an acceleration and therefore a velocity and a position. From this point, a new density can be computed and a new time step can be started.

These five steps have been adapted for GPUs and, more important, the entire sequence has been integrated for GPUs in order to limit performance losses due to data transfer from/to the CPU or main memory. That means that, as soon as the input data (initial positions and velocities) are calculated or read from a file, the data is sent to the GPU *once and for all* and needs to be brought back on the host only to be written to the output file.

3 Performances

3.1 Setup of the Experiments

The subsequent experiments were performed on three sets of initial conditions. First a Plummer sphere [13] where the particles are distributed isotropically and satisfy a well-defined profile. The velocity distribution is chosen in order to balance the gravity and the sphere remains coherent over long period of time. Second, we simulated an exponential disk, where particles are distributed in a thin plane and velocities are similar to the one measured in real disc-like galaxies. This experiment is set up to be unstable by nature, allowing spiral arms to develop for instance. The third type of simulations consists simply in particles randomly distributed in a cubic space with random velocities. It is similar to cosmological simulations that are ignited from a quasi-uniform distribution of matter. All the simulations are performed on the same volume using 32^3, 64^3 and 128^3 particles/density cells : smaller problems are of little interests while the next power of 8 (256^3) surpasses the current capacity some routines such as CUFFT and CUDPP sort and compact. These larger situations will be addressed on short term as hardware and software improve and our set of simulation already provides a good insight on the perspectives offered by GPUs. The time step is chosen in order to achieve an energy conservation of $\Delta E/E \sim 10^{-3}$ over a time unit, where the energy is defined as $E = \sum_{\text{particles}} \frac{v^2 - \phi(x)}{2}$. The tests were run on a single GeForce 8800 GTX device.

For comparison, we developed a CPU version of the PM integrator, written in C, compiled using the Intel C compiler. CPU-driven simulations were performed on Opterons Dual-Core at 2.2 GHz, which also hosts the GPU we used. Let us emphasize that *by no means* the CPU version should be considered as fully optimized version, even though loops were optimized by the compiler. Comparing with the PM sequence of existing codes (like Gadget or Ramses [1,3]) might be irrelevant because of critical algorithm differences. The following results should be more considered as a demonstration of the quick gains that can be achieved on GPUs with moderate coding skills.

On the CPU, Fourier transforms were performed with the FFTW 3.1.2 library using the single-float precision version and multi-threading was not enabled (as in e.g. [14] used hereafter as a comparison). On both the GPU and CPU, FFTs were performed using complex-to-complex transform. The multi-grid calculation involved 3 V-cycles with five levels of restrictions, using 5 pre- and post-recursion smoothing steps. It ensures the same level of energy conservation as the FFT calculation $\Delta E/E \sim 10^{-3}$. The energy fluctuations were found to be identical between the CPU and GPU versions, even though not all floating point operations are IEEE compliant on GPU.

Each of the steps of the integrator is timed separately and ran 1000 times. As explained in section 2, we used two different histogramming procedures, one that involves a partial CPU calculation ("GPU Mixed Histo" hereafter) and one "Full GPU" version that suppresses all CPU calculations at the cost of complex sorting and compact routines.

3.2 Overall Performance

According to our experiments, the GPU versions of the PM integrator can be significantly accelerated, depending of the type of simulation and the number of cells. As a first broad picture, the figure 2 presents the overall speedup of Full-GPU and GPU-Mixed Histo calculations compared to the CPU version for 32^3, 64^3 and 128^3 simulations. Using FFTs for the Poisson equations, the gain measured for the Full GPU ranges from a factor 1.6 to 11.5 while the mixed version experiences a speedup from 2.5 for the smallest versions to 18 for the

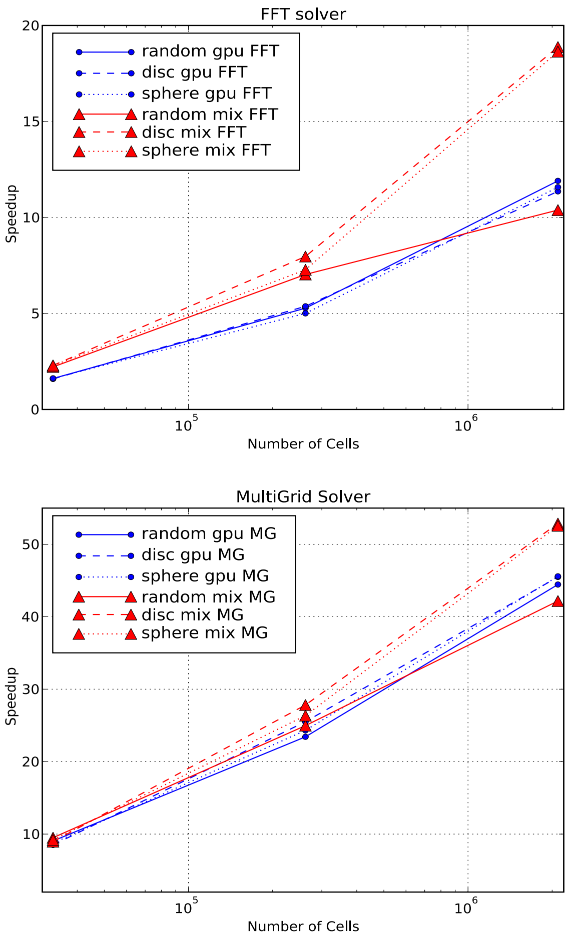


Fig. 2. The overall speedup compared to the CPU version using an FFT (top panel) or a multi-grid (bottom panel) solver, shown as a function of the number of cells. Please note that y-scales are different in the two panels. Blue lines with circular symbols stand for the full-GPU calculations, while red ones with diamonds stand for the GPU simulations with the histogramming partially performed on the host.

largest simulation. Using the multi-grid relaxation, the speedup is at least a factor 10 for both versions of the histogramming and reaches a level of 43 for the Full-GPU versions and 52 for the GPU-Mix 128^3 calculations. Focusing on the 128^3 simulations, the GPU-mixed versions are almost twice faster than the GPU versions, while Multi-Grid based calculations are almost equivalent if one considers the Full-GPU or GPU-mixed calculations. One can also note that the random calculations are less efficient when the histogram is performed on the CPU. All these trends result from different limitations and specificities that are detailed hereafter.

3.3 Bottlenecks

CPU and GPU versions are limited by different bottlenecks. It can be seen from Fig. 3 which details the execution times of the different steps involved in a single timestep for the 128^3 simulations. Histogramming on the GPU and Mixed histogramming are shown side by side even if only one of these implementations is used in a given simulation run. The same remark holds for the FFT-based and the MG-based Poisson solvers.

The CPU version depends heavily on the performance of the Poisson solver: about 75 % of a timestep is spent solving the Poisson equation when the FFT solver is called while this proportion is close to a 100% when the MG solver is used.

Meanwhile, the Full-GPU version is more influenced by the density calculation. Histogramming is not naturally suited to SIMD calculations and involves

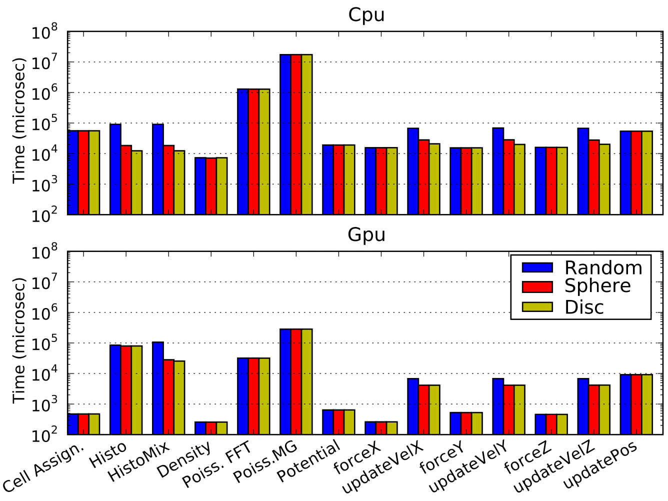


Fig. 3. Absolute timings for the different stages of a single time step for the CPU and the GPU versions for 128^3 simulations. For a given run, histogramming is performed either on the GPU (Histo) or on the CPU (HistoMix). Also the Poisson equation is solved using FFT or Multi-Grid relaxation (MG). For the CPU version, the Histo and HistoMix routines are identical.

complex operations (sorting/segmented scan). If FFT is used for the Poisson equation, 60 % of its computational time is spent constructing the density. Using MG, the fraction of time spent in the histogramming is lowered to levels of 10% to 20%, but only because MG is less efficient than FFT.

Finally, the Mixed-Histo calculation manages to divide by two the duration of histogramming in the overall calculation compared to the full GPU even though costly transfers are involved (except for the random distribution, see section 3.4). In FFT-based simulations, the same amount of time is then spent in the Poisson solver and in the histogramming. It explains the significant increase of performance observed in the overall calculation when switching to this type of histogram. In MG-based calculation, the weight of the histogramming becomes even less important than it was. A moderate increase in perfomance is then observed, as reported in figure 2.

3.4 Sub Components Analysis

In addition to the absolute timings in Fig. 3, the speedups (compared to the CPU versions) of the sub components of a time step are shown in Fig. 4. From these measurments, we deduce that the overall speedup of the GPU version compared to the CPU results mainly from a large gain in the resolution of the Poisson equation, moderated by the low efficiency of the histogramming and sustained by the speedups achieved in all the other steps of the calculation. The analysis of the individual sub components follow.

Poisson solver. From Fig. 3, it can be noted that the resolutions of the Poisson equations are extremely time consuming for the CPU versions and among them the multi-grid calculations are ten times slower than FFTs. The same difference can be noted on the GPU versions, even though the speedup on GPU is 40 (for the FFT) or 60 (for the Multi-Grid). Let us emphasize that the FFT-driven resolution of the Poisson equation involves two Fourier transforms in opposite directions and an isotropic filtering. If we consider only the FFTs, our measurements showed that CUFFT is 2, 16 and 40 times faster than FFTW for the 32^3, 64^3 and 128^3 experiments. It differs from measurements of [15] with a different GPU but are consistent with the tests of [14].

Important speedups are measured for the multi-grid relaxation, where both GPU and CPU code were written from scratch. Speedup is achieved using the high-level parallelism of the computations involved in the restrictions, prolongations and the Gauss-Seidel iterations. We think greater speedups might be reached by fine-tuning the GPU routines, especially with a greater use of shared memory for the redundant operations of restrictions/prolongation.

Histogramming. On the downside, no gain can be observed for the histogramming step on the GPUs. Even worse, this computation can be 5 times slower on 128^3 simulations an can go down to 10 times slower on the full GPU version for 32^3 particles simulations (not studied here otherwise). The GPU-mix histoversion performs the most expensive step on the host but still, the data transfer

(transfer rate and latency) results in this computation step being twice slower than computation performed only on the CPU. The mixed version fill the gap for the sphere and disc simulations, but are at least 15% slower than the CPU versions.

It should be mentionned that higher order assignment scheme exist (Cloud-In-Cell, Triangular-Shaped-Cloud) where particles contribute to more than one cell [6], resulting in the calculation of 8(CIC) or 27 (TSC) histograms per timestep. The CPU or mixed-GPU may suffer from these successive calculations. Conversely, the algorithm described by [8] requires only one sort and may become more competitive as higher order (and more accurate) assignement schemes are used.

Interestingly, the random case simulations are less favorable to the CPU versions in terms of histogramming: it cannot be as efficient as it writes data in memory due to the fact that particles are spread in all the computation box, presumably due to cache misses. In the two previous cases, the particles were confined in certain sub regions (disc or sphere), ensuring a certain level of cache hits. Meanwhile, the SIMD GPU histogramming routine relies on sort/scan primitives which do not depend strongly on the particle distribution.

Accelerations and updates. All the other stages of the calculation are significantly speeded up on GPU, with speedups ranging from 5 to 120 compared to the CPU versions.

We noticed that the speedups of velocity updates increase as the particles are spread in a larger portion of the grid, especially comparing disc simulations to random simulations. To compute the velocity of a particle, a given thread (or the CPU) finds the cell it belongs to and uses the associated acceleration. Consequently, CPU is less efficient to access memory in a random fashion as a larger fraction of the grid is occupied by particles (see also Fig. 3): it results

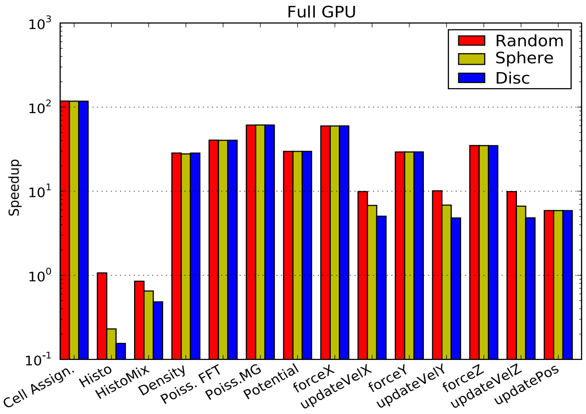


Fig. 4. Speedup for the different stages of a single time step for the the GPU-Mix and full GPU simulations on a 128^3 grid

in a greater speedup in favor of the GPU. Conversely, if particles are strongly clustered, memory access scheme gets close to neighboring accesses to the memory, which are more likely to be cached on the CPU.

Finally, it should be noted that the force/acceleration computation is more efficient along the x direction. It results from the storage of the grids in memory which favors certain directions in terms of contiguous memory access among threads.

4 Conclusion and Perspectives

Using CUDA API we developed a Particle-Mesh N-Body integrator that runs on common graphic devices. All the steps of the algorithm were parallelized and we obtain speedups that ranges from 2 to 50 depending on the size of the problem and the choice of techniques. The density computation and the Poisson solver are the critical part of the implementation. We implemented a full GPU histogramming algorithm and solve the Poisson equation by means of FFT and MG relaxation. For large problems, the resolution of the Poisson equation is at least 40 times faster on the GPU using FFT and 60 times faster using multi-grid relaxation, while the histogram computation is hardly accelerated on GPU. Combined with the 10-100 speedups obtained on all the other steps (cell assignment, acceleration computation/interpolation, velocity/position update) a significant acceleration of the code is observed : for a typical 128^3 data set we achieve speedups of 12 for FFT based calculations and up to 45 for MG based ones. If the histogramming is partially performed on the host, higher speedups of 20 for FFT-based and 50 for MG-based simulations are observed.

In a near future, we plan first to assess larger problems (256^3, 512^3) in order to reach astrophysically relevant resolutions. Using devices with larger memory capabilities and supporting atomic operations, it should be within our reach using the Multi-Grid Poisson solver and an improved version of the histogramming routine (following e.g. [16]). If such large situations can be handled, running large multi-GPU simulations would be the next step to reach the billion particles with GPU speedups. However, it remains still unclear how communications between GPUs through the hosts may lower the speedups obtained with a single device.

From an astrophysical point of view, the results obtained are encouraging. Furthermore, this PM development is accompanied by two other codes : GPU version of a non-linear FAS Multi-Grid solver for the modified Newtonian dynamics and the CUDA transcription of a cosmological radiative transfer code. For these two other codes, speedups range from 10 to 80 compared to CPU versions. It opens interesting perspectives of an easy access to HPC-like calculations in their desktop machines with a set of well-suited API's enabled for GPUs.

Acknowledgments

D.A. would like to thank R. Teyssier for fruitful comments and T.Keller for technical support. This work is supported by a grant from the *Conseil Scientifique* de l'Université Louis Pasteur.

References

1. Springel, V., et al.: Simulations of the formation, evolution and clustering of galaxies and quasars. Nature 435, 629–636 (2005)
2. Iliev, I.T., et al.: Simulating Cosmic Reionization. ArXiv e-prints 806 (June 2008)
3. Teyssier, R., et al.: Full-Sky Weak Lensing Simulation with 70 Billion Particles. ArXiv e-prints 807 (July 2008)
4. Portegies Zwart, S.F., Belleman, R.G., Geldof, P.M.: High-performance direct gravitational N-body simulations on graphics processing units. New Astronomy 12, 641–650 (2007)
5. Hamada, T., Iitaka, T.: The Chamomile Scheme: An Optimized Algorithm for N-body simulations on Programmable Graphics Processing Units. ArXiv Astrophysics e-prints (March 2007)
6. Hockney, R.W., Eastwood, J.W.: Computer simulation using particles. Hilger, Bristol (1988)
7. Bertschinger, E.: Simulations of Structure Formation in the Universe. ARAA 36, 599–654 (1998)
8. Ferrell, R., Bertschinger, E.: Particle-Mesh Methods on the Connection Machine. Contributions to Mineralogy and Petrology, 10002–+ (November 1993)
9. Press, W.H.: Numerical recipes in C++: the art of scientific computing (2002)
10. Brada, R., Milgrom, M.: Stability of Disk Galaxies in the Modified Dynamics. ApJ 519, 590–598 (1999)
11. Sorensen, T., Schaeffter, T., Noe, K., Hansen, M.: Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. IEEE Transactions on Medical Imaging 27(4), 538–547 (2008)
12. Göddeke, D., Strzodka, R., Turek, S.: Performance and accuracy of hardware-oriented native-, emulated-and mixed-precision solvers in fem simulations. Int. J. Parallel Emerg. Distrib. Syst. 22(4), 221–256 (2007)
13. Binney, J., Tremaine, S.: Galactic Dynamics, 2nd edn. (2008)
14. Demores, P.: GPU Benchmarking (2007), `www.cv.nrao.edu/pdemores/gpu/`
15. Merz, H.: CUFFT Vs FFTW Comparison (2008), `http://www.science.uwaterloo.ca/~hmerz/CUDAbenchFFT/`
16. Stantchev, G., Dorland, W., Gumerov, N.: Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu. Journal of Parallel and Distributed Computing 68(10), 1339–1349 (2008)

A Note on Auto-tuning GEMM for GPUs

Yinan Li[1], Jack Dongarra[1,2,3], and Stanimire Tomov[1]

[1] University of Tennessee, USA
[2] Oak Ridge National Laboratory, USA
[3] University of Manchester, UK

Abstract. The development of high performance dense linear algebra (DLA) critically depends on highly optimized BLAS, and especially on the matrix multiplication routine (GEMM). This is especially true for Graphics Processing Units (GPUs), as evidenced by recently published results on DLA for GPUs that rely on highly optimized GEMM. However, the current best GEMM performance, e.g. of up to 375 GFlop/s in single precision and of up to 75 GFlop/s in double precision arithmetic on NVIDIA's GTX 280, is difficult to achieve. The development involves extensive GPU knowledge and even backward engineering to understand some undocumented insides about the architecture that have been of key importance in the development. In this paper, we describe some GPU GEMM **auto-tuning** optimization techniques that allow us to keep up with changing hardware by rapidly reusing, rather than reinventing, the existing ideas. Auto-tuning, as we show in this paper, is a very practical solution where in addition to getting an easy portability, we can often get substantial speedups even on current GPUs (e.g. up to 27% in certain cases for both single and double precision GEMMs on the GTX 280).

Keywords: Auto-tuning, matrix multiply, dense linear algebra, GPUs.

1 Introduction

Recent activities of major chip manufacturers, such as Intel, AMD, IBM and NVIDIA, make it more evident than ever that future designs of microprocessors and large HPC systems will be hybrid/heterogeneous in nature, relying on the integration (in varying proportions) of two major types of components:

1. Multi/many-cores CPU technology, where the number of cores will continue to escalate while avoiding the power wall, instruction level parallelism wall, and the memory wall [2]; and
2. Special purpose hardware and accelerators, especially GPUs, which are in commodity production, have outpaced standard CPUs in performance, and have become as easy, if not easier to program than multicore CPUs.

The relative balance between these component types in future designs is not clear, and will likely vary over time, but there seems to be no doubt that future generations of computer systems, ranging from laptops to supercomputers, will consist of a composition of heterogeneous components.

These hardware trends have inevitably brought up the need for updates on existing legacy software packages, such as the sequential LAPACK [1], from the area of DLA. To take advantage of the new computational environment, our current research shows that successors of LAPACK have to incorporate algorithms of three main characteristics: **high parallelism** (to efficiently account for the many-cores available), **reduced communication** (to account for the exponentially increasing memory wall), and **heterogeneity-awareness** (meaning, algorithms to be properly split between the components of the heterogeneous system so that the strengths of each component are properly matched to the requirement of the algorithm). This is reflected for example in the *Matrix Algebra on GPU and Multicore Architectures* (MAGMA) project [3], a recent effort on developing a successor to LAPACK but for heterogeneous/hybrid architectures, with current stress on Multicore + GPU systems.

Our overall goals, as related to auto-tuning and the MAGMA project, are to investigate opportunities for automating the transition to MAGMA and moreover, automating the tuning process of the newly discovered algorithms, both for the sake of productivity and for correctness in the new, complex, and rapidly changing computational environment. However, the techniques developed and incorporated in MAGMA so far show that a transition from LAPACK to MAGMA can not be done automatically or with minor modifications, as in many cases new algorithms that significantly differ from algorithms for conventional architectures will be needed [3]. Indeed, experiments show that the easy approach, that has been successful in the past, to use current LAPACK and simply call BLAS on the GPU leads to significant loss of performance (that can be of order $3\times$ and higher). Nevertheless, this approach can lead to high performance, but only after some modifications and for routines that map well on the GPU, like Cholesky (e.g. Dongarra et al. [8] report up to 327 GFlop/s in single precision on a pre-released at the time NVIDIA T10P). Naturally, previous attempts to wrap some of the work needed in transitions like this in frameworks, have also failed to produce convincing results. For example the FLAME project [10], is a framework to facilitate the implementation of a class of DLA algorithms. Originally designed before the appearance of multicores, had to be continuously updated to meet the challenges of emerging architectures. Still, in the context of GPUs in particular, the latest results from FLAME developers show a single precision Cholesky factorization running at up to 156.2 GFlop/s, and a single precision LU factorization at up to 142 GFlop/s [4]. Although impressive, compare this performance with, accordingly, 315 GFlop/s and 309 GFlop/s [11] for the new algorithms (all these results are for the GTX 280). The main point here is that emerging architectures have motivated the development of new algorithms that have a much larger design space than previously needed. Early autotuners for example only targeted the BLAS, under the assumption that a few parameters (e.g. block sizes) were enough to capture enough of the algorithmic design space of higher level algorithms (LU, etc.) to attain a large fraction of peak performance. This assumption was adequate to keep LAPACK and ScaLAPACK reasonably efficient for many years, but as described above it is far from adequate going forward. This is even more true for frameworks that are rooted in the old sequential environments preceding the introduction of multicores.

This brief outline motivates us to use auto-tuning (as a major component of the new MAGMA efforts) to keep up with the rapidly innovating hardware and continually growing design space so that we get to rapidly reuse, rather than reinvent, the new ideas. Indeed, the work that we describe in this paper on developing GEMM autotuners shows that we can significantly accelerate current results not only on emerging GPUs (e.g. when GPUs recently added support for double precision arithmetic) but also on current GPUs for which the algorithms were originally designed. Moreover, we have discovered that in the new hardware environment our design spaces critically depend not only on the architecture but also on problem sizes. The implication is that there may be different optimal algorithms for the same problem, and discovering these algorithms and their tuning on a case by case study may be impractical even for an expert. Auto-tuning is preferable.

The rest of the paper is organized as follows. In Section 2, we give background information on auto-tuning for DLA. Section 3 describe our GEMM autotuner for GPUs. Next are performance results (Section 4) and finally conclusions and future directions (Section 5).

2 Auto-tuning for CPUs

Automatic performance tuning (optimization), or auto-tuning in short, is a technique that has been used intensively on CPUs to automatically generate near-optimal numerical libraries. For example, ATLAS [12,7] and PHiPAC [5] are used to generate highly optimized BLAS. In addition, FFTW [9] is successfully used to generate optimized libraries for FFT, which is one of the most important techniques for digital signal processing. There are generally two kinds of approaches for doing auto-tuning, specifically model-driven optimization and empirical optimization. The idea of model-driven optimization comes from the compiler community. The compiler community has developed various optimization techniques that can be effectively used to transform code written in high-level languages such as C and Fortran to run efficiently on modern CPU architectures. These optimization techniques include loop blocking, loop unrolling, loop permutation, fusion and distribution, prefetching, and software pipelining. The parameters for these transformations such as the block size and the amount of unrolling are determined by analytical models, which are commonly used in the compiler community. While model-driven optimization is generally effective to make programs run faster, it may not give optimal performance to special-purpose libraries for linear algebra and signal processing. The reason is that analytical models used by compilers are only simplified abstractions of the underlying processor architectures, and they must be general enough to be applicable to all kinds of programs. Thus, the limited accuracy of analytical models makes the model-driven approach not so attractive for the optimization of highly special kernels for linear algebra and signal processing, if the approach is solely used. In contrast to model-driven optimization, empirical optimization techniques generate a large number of parametrized code variants for a given algorithm and run these variants on a given platform to discover the one that gives the best performance. The effectiveness of empirical optimization depends on the chosen parameters to optimize, and the search heuristic used. A disadvantage of empirical optimization is the time cost of searching for the best code variant, which is usually proportional to the number of variants generated and evaluated. Contrarily, model-driven optimization has a O(1) cost, since the parameters can be derived from the analytical model. Therefore, a natural idea is to combine these two approaches, and it gives the third approach, a hybrid approach that uses the model-driven approach in the first stage to limit the search space for the second stage of empirical search.

Another aspect of the auto-tuning, besides the compiler and empirical tuning where an optimal computational kernel is generated as it is installed on one system, is **adaptivity** which can be regarded in various aspects [6]. The main aspect is to treat cases where tuning can not be restricted to optimizations at design time, installation time, or even compile time. In those cases, mechanisms of adaptivity can be incorporated in software, where tuning information captured in prior runs can be used to tune future runs.

With the success of auto-tuning techniques on generating highly optimized DLA kernels on CPUs, it is interesting to see how the idea can be used to generate near-optimal DLA kernels on modern high-performance GPUs.

3 GEMM Autotuner for GPUs

In this section we present our preliminary study on the idea of auto-tuning on modern GPUs. In particular, we design a GEMM "autotuner" for NVIDIA CUDA-enabled GPUs. Here autotuner refers to an auto-tuning system that automatically generates and searches a space of algorithms.

There are two core components in a complete auto-tuning system: a code generator and a heuristic search engine. The code generator generates parametrized code variants according to a pre-defined code template. The heuristic search engine then runs these variants and finds out the best one using a feedback loop, i.e., the performance results of previously evaluated variants are used as a guidance for the search on currently unevaluated variants.

In [11], Volkov and Demmel present kernels for single-precision matrix multiplication (SGEMM) that significantly outperformed CUBLAS 1.0 on CUDA-enabled GPUs, using an approach that challenges those optimization strategies and programming guidelines that are commonly accepted. In this paper, we focus on the GEMM kernel that computes $C = \alpha A \times B + \beta C$. Additionally, we investigate auto-tuning on both single precision and double precision GEMM kernels (i.e., SGEMM and DGEMM). The SGEMM kernel proposed in [11] takes advantage of the vector capability of NVIDIA CUDA-enabled GPUs. The

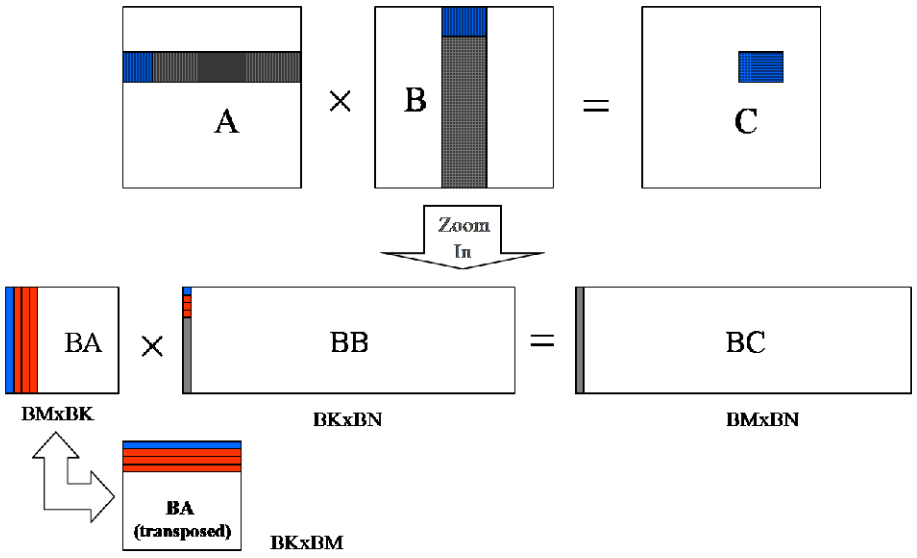


Fig. 1. The algorithmic view of the code template for GEMM

authors argue that modern GPUs should be viewed as multi-threaded vector units, and their algorithms for matrix multiplication resemble those earlier ones developed for vector processors. We take their SGEMM kernel for computing $C = \alpha A \times B + \beta C$ as our code template, with modifications to make the template accept row-major input matrices, instead of column major used in their original kernel.

Figure 1 depicts the algorithmic view of the code templates respectively for both SGEMM and DGEMM. Suppose A, B, and C are M×K, K×N, and M×N matrices, and that M, N, and K are correspondingly divisible by BM, BN, and BK (otherwise "padding" by zero has to be applied or using the host for part of the computation). Then the matrices A, B, and C are partitioned into blocks of sizes BM×BK, BK×BN, and BM×BN, respectively (as illustrated on the figure). The elements of each BM×BN block of the matrix C (denoted by BC on the figure, standing for 'block of C') are computed by a $t_x \times t_y$ thread block. Depending on the number of threads in each thread block, each thread will compute either an entire column or part of a column of BC. For example, suppose BM = 16 and BN = 64, and the thread block has 16 × 4 threads, then each thread will compute exactly one column of BC. If the thread block has 16 × 8 threads, then each thread will compute half of a column of BC. After each thread finishes its assigned portion of the computation, it writes the results (i.e., an entire column or part of a column of BC back to the global memory where the matrix C resides. In each iteration, a BM×BK block BA of the matrix A is brought into the on-chip shared memory and kept there until the computation of BC is finished. Similarly to the matrix C, matrix B always resides in the global memory, and the elements of each block BB are brought from the global memory to the on-chip registers as necessary in each iteration. Because modern GPUs have a large register file within each multiprocessor, a significant amount of the computation can be done in registers. This is critical to achieving near-optimal performance. As in [11], the computation of each block BC = BC + BA×BB is fully unrolled. It is also worth pointing out that in our SGEMM, 4 `saxpy` calls and 4 memory accesses to BB are grouped together, as in [11], while in our DGEMM, each group contains 2 `saxpy` and 2 memory accesses to BB. This is critical to achieving maximum utilization of memory bandwidth in both cases, considering that the different widths between `float` and `double`.

As outlined above, 5 parameters (BM, BK, BN, t_x, and t_y) determine the actual implementation of the code template. There is one additional parameter that is of interest to the actual implementation. This additional parameter determines the layout of each block BA of the matrix A in the shared memory, i.e., whether the copy of each block BA in the shared memory is transposed or not. Since the share memory is divided into banks and two or more simultaneous accesses to the same bank cause the so-called bank conflicts, transposing the layout of each block BA in the shared memory may help reduce the possibility of bank conflicts, thus potentially improving the performance. Therefore, the actual implementation of the above code template is determined or parametrized by 6 parameters, namely BM, BK, BN, t_x, t_y, and a flag *trans* indicating whether to transpose the copy of each block BA in the shared memory.

We implemented code generators for both SGEMM and DGEMM on NVIDIA CUDA-enabled GPUs. The code generator takes the 6 parameters as inputs, and generates the kernel, the timing utilities, the header file, and the Makefile to build the kernel. The code generator first checks the validity of the input parameters before actually generating the files. By validity we mean 1) the input parameters confirm to hardware constraints, e.g., the maximum number of threads per thread block $t_x \times t_y \leq 512$, and 2) the input parameters are mutually compatible, e.g., $(t_x \times t_y)\%\text{BK} = 0$, $\text{BM}\%t_y = 0$, and $\text{BN}\%t_x = 0$. By varying the input parameters, we can generate different variants of the kernel, and evaluate their performance, in order to identify the best variant. One way to implement auto-tuning is to generate a small number of variants for some matrices with typical sizes during installation time, and choose the best variant during run time, depending on the input matrix size.

4 Performance Results

The performance results in this section are for NVIDIA's GeForce GTX 280.

First, we evaluate the performance of the GEMM autotuner in both single and double precision. Figure 2, Left compares the performance of the GEMM autotuner in single precision with the CUBLAS 2.0 SGEMM for multiplying square matrices. We note that both CUBLAS 2.0 SGEMM and our auto-tuned SGEMM are based on V.Volkov's SGEMM [11]. The GEMM autotuner selects

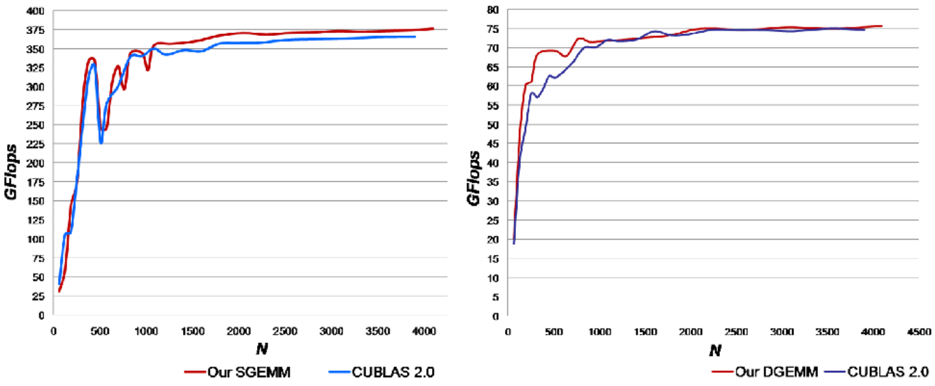


Fig. 2. Performance comparison of CUBLAS 2.0 *vs* auto-tuned SGEMM (left) and DGEMM (right) on square matrices

the best performing one among several variants. It can be seen that the performance of the autotuner is apparently slightly better than the CUBLAS 2.0 SGEMM. Figure 2, Right shows that the autotuner also performs better than CUBLAS in double precision. These preliminary results demonstrate that auto-tuning is promising in automatically producing near-optimal GEMM kernels on GPUs. The most attractive feature of auto-tuning is that it allows us to keep up

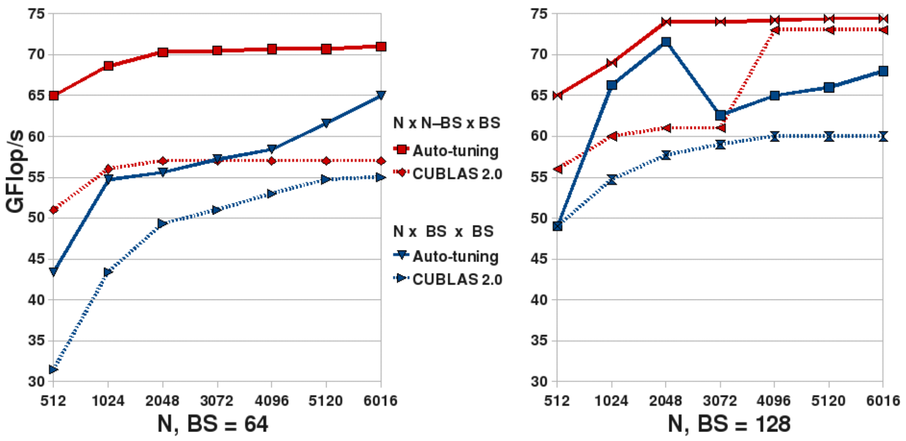


Fig. 3. Performance comparison of the auto-tuned (solid line) *vs* CUBLAS 2.0 (dotted line) DGEMMs occurring in the block LU factorization (for block sizes BS = 64 on the left and 128 on the right) of a matrix of size 6144 × 6144. The two kernels shown are for multiplying N×BS and BS×N−BS matrices (denoted by N×N−BS×BS), and N×BS and BS×BS matrices (denoted by N×BS×BS)

with changing hardware by automatically and rapidly generating near-optimal BLAS kernels, given any newly developed GPUs.

The fact that the two performances are so close is not surprising because our auto-tuned code and CUBLAS 2.0's code are based on the same kernel, and this kernel was designed and tuned for current GPUs (and in particular the GTX 280), targeting high performance for large matrices. In practice though, and in particular in developing DLA algorithms, it is very important to have high performance GEMMs on rectangular matrices, where one size is large, and the other is fixed within a certain block size (BS), e.g. BS = 64, 128, up to about 256 on current architectures. For example, in an LU factorization (with look-ahead) we need two types of GEMM, namely one for multiplying matrices of size N×BS and BS×N−BS, and another for multiplying N×BS and BS×BS matrices. This situation is illustrated on Figure 3, where we compare the performances of the CUBLAS 2.0 *vs* auto-tuned DGEMMs occurring in the block LU factorization of a matrix of size 6144×6144. The graphs show that our auto-tuned code significantly outperforms (up to 27%) the DGEMM from CUBLAS 2.0.

These results support experiences and observations by others on "how sensitive the performance of GPU is to the formulations of your kernel" [13] and that an enormous amount of well thought experimentation and benchmarking [11,13] is needed in order to optimize performance.

5 Conclusions and Future Directions

We highlighted the difficulty in developing highly optimized codes for new architectures, and in particular GEMM for GPUs. On the other side, we have shown an auto-tuning approach that is very practical and can lead to optimal performance. In particular, our auto-tuning approach allowed us

– To easily port existing ideas on quickly evolving architectures (e.g. demonstrated here by transferring single precision to double precision GEMM designs for GPUs), and
– To substantially speed up even highly tuned kernels (e.g. up to 27% in this particular study).

These results also underline the need to incorporate auto-tuning ideas in our software. This is especially needed now for the new, complex, and rapidly changing computational environment. Therefore our future directions are, as we develop new algorithms (e.g. within the MAGMA project), to systematically define their design/search space, so that we can easily automate the tuning process as demonstrated in this paper.

Acknowledgments. Part of this work was supported by the U.S. National Science Foundation, and the U.S. Department of Energy. We thank NVIDIA and NVIDIA's Professor Partnership Program for their hardware donations.

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK user's guide, 3rd edn. SIAM, Philadelphia (1999)
2. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., Yelick, K.A.: The landscape of parallel computing research: A view from berkeley, Tech. Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (December 2006)
3. Baboulin, M., Demmel, J., Dongarra, J., Tomov, S., Volkov, V.: Enhancing the performance of dense linear algebra solvers on GPUs [in the MAGMA project]. Poster at Supercomputing 2008, November 18 (2008), http://www.cs.utk.edu/~tomov/SC08-poster.pdf
4. Barrachina, S., Castillo, M., Igual, F., Mayo, R., Quintana-Orti, E., Quintana-Orti, G.: Exploiting the capabilities of modern GPUs for dense matrix computations, Technical Report ICC 01-11-2008, Universidad Jaime I, Spain (2008)
5. Bilmes, J., Asanovic, K., Chin, C.-W., Demmel, J.: Optimizing Matrix Multiply Using PHiPAC: A Portable, High-Performance, ANSI C Coding Methodology. In: International Conference on Supercomputing, pp. 340–347 (1997)
6. Bosilca, G., Chen, Z., Dongarra, J., Eijkhout, V., Fagg, G., Fuentes, E., Langou, J., Luszczek, P., Pjesivac-Grbovic, J., Seymour, K., You, H., Vadiyar, S.S.: Self adapting numerical software (SANS) effort. IBM Journal of Reseach and Development 50(2/3), 223–238 (2006)
7. Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, C., Yelick, K.: Self adapting linear algebra algorithms and software. Proceedings of the IEEE 93(2) (2005); special issue on Program Generation, Optimization, and Adaptation
8. Dongarra, J., Moore, S., Peterson, G., Tomov, S., Allred, J., Natoli, V., Richie, D.: Exploring new architectures in accelerating CFD for Air Force applications. In: Proceedings of HPCMP Users Group Conference 2008, July 14-17 (2008), http://www.cs.utk.edu/~tomov/ugc2008_final.pdf
9. Frigo, M., Johnson, S.G.: FFTW: An Adaptive Software Architecture for the FFT. In: Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing, vol. 3, pp. 1381–1384. IEEE, Los Alamitos (1998)
10. Gunnels, J.A., Van De Geijn, R.A., Henry, G.M.: Flame: Formal linear algebra methods environment. ACM Transactions on Mathematical Software 27, 422–455 (2001)
11. Volkov, V., Demmel, J.: Benchmarking GPUs to tune dense linear algebra. In: Supercomputing 2008. IEEE, Los Alamitos (2008) (to appear)
12. Whaley, R.C., Petitet, A., Dongarra, J.: Automated Empirical Optimizations of Software and the ATLAS Project. Parallel Computing 27(1-2), 3–35 (2001)
13. Wolfe, M.: Compilers and More: Optimizing GPU Kernels, 10/2008, HPC Wire, http://www.hpcwire.com/features/33607434.html

Fast Conjugate Gradients with Multiple GPUs

Ali Cevahir[1], Akira Nukada[1], and Satoshi Matsuoka[1,2]

[1] Tokyo Institute of Technology,
{ali,nukada}@matsulab.is.titech.ac.jp, matsu@is.titech.ac.jp
[2] National Institute of Informatics, Japan

Abstract. The limiting factor for efficiency of sparse linear solvers is the memory bandwidth. In this work, we describe a fast Conjugate Gradient solver for unstructured problems, which runs on multiple GPUs installed on a single mainboard. The solver achieves double precision accuracy with single precision GPUs, using a mixed precision iterative refinement algorithm. To achieve high computation speed, we propose a fast sparse matrix-vector multiplication algorithm, which is the core operation of iterative solvers. The proposed multiplication algorithm efficiently utilizes GPU resources via caching, coalesced memory accesses and load balance between running threads. Experiments on wide range of matrices show that our matrix-vector multiplication algorithm achieves up to 11.6 Gflops on single GeForce 8800 GTS card and CG implementation achieves up to 24.6 Gflops with four GPUs.

1 Introduction

Recently, GPUs have attracted HPC community, because of their peak compute capability and high memory bandwidth, compared to conventional CPUs. Moreover, today's GPUs achieve relatively small cost and power consumption vs. their performance. APIs developed by manufacturers like CTM [1] and CUDA [2] made GPUs easy to program as a highly parallel multi-core coprocessor, not only for graphic applications but also for non-graphic applications.

Several advantages of GPU computing is mentioned above. On the other hand, it is not easy to achieve high utilization of GPU resources for the solution of unstructured sparse linear systems. The performance of the sparse matrix-vector multiplication (MxV), which is in the core of sparse linear iterative solvers, is limited by the memory bandwidth rather than peak computation power. Although new generation GPUs are capable of being programmed for general purpose computations, they are originally optimized for graphics applications. Therefore, to achieve high performance, thread level parallelism and memory access methods on a set of streaming multi-processors should be carefully thought.

Another drawback of GPUs is, for most of them, lack of double precision support for floating point operations. Hence, solvers without CPU support suffer from loss of accuracy in solution. In order to achieve double precision accuracy, we adopt a mixed precision iterative refinement algorithm [6].

In this work, we implement a fast Conjugate Gradient (CG) solver with multi-GPU support. To the best of our knowledge, this is the first multi-GPU solver

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 893–903, 2009.

for unstructured sparse systems. All basic operations of the single precision CG solver are implemented on GPUs. We propose a fast MxV algorithm.Proposed MxV algorithm achieves high utilization of GPU resources by coalesced memory accesses, caching, and load balancing between working threads.

We evaluate performance of the proposed algorithms over a wide range of well-known matrices. We compare the performance of our MxV algorithm on the GPU with a CPU implementation and two naïve GPU implementations. Experiments confirm that our algorithm on the GPU is several times faster than other implementations. Performance of the parallel solver degrades to some extent due to the communication between GPUs and the CPU in each iteration. Still, we achieve average speedup of 2.83 over single GPU on 4 GPUs for big matrices in the dataset. Although we have reported results only for CG algorithm in this work, our approaches can be efficiently applied for other sparse linear methods with GPU support.

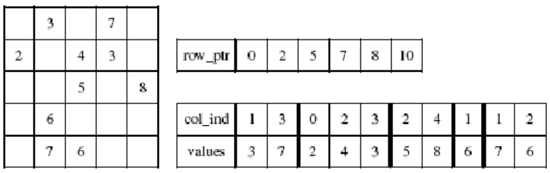
2 Background

2.1 Sparse Matrix Storage Formats

Since many of the entries in sparse matrices are zero, there is no need to explicitly store them. There are many compressed storage formats for sparse matrices [17]. In this section, we only mention two of them related to our work.

Compressed Storage by Rows (CSR) stores nonzeros of the matrices in row order. For indexing nonzeros, two arrays are used. The elements in the row pointer array point the first nonzero in each row. There are number of rows + 1 elements in this array where the last element is kept for indicating boundary of the last row. The other array stores the column indices of the nonzeros in row order. Fig. 1 depicts an example for CSR and the pseudocode of MxV ($y = Ax$) for a CSR-stored matrix A with n rows. In the example, row_ptr and col_ind stand for row pointers and column indices, respectively.

JDS format is not as straightforward as CSR. It can result with better performance for MxV on vector processors. In order to store a matrix in JDS format,



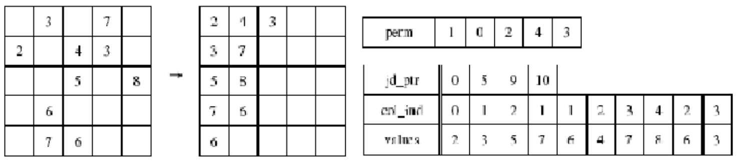
	3		7	
2		4	3	
		5		8
	6			
	7	6		

row_ptr	0	2	5	7	8	10

col_ind	1	3	0	2	3	2	4	1	1	2
values	3	7	2	4	3	5	8	6	7	6

```
for $i \leftarrow 0$ to $n - 1$ do
    $y[i] \leftarrow 0$
    for $j \leftarrow row\_ptr[i]$ to $row\_ptr[i + 1] - 1$ do
        $y[i] \leftarrow y[i] + values[j] \times x[col\_ind[j]]$
```

Fig. 1. A CSR example and pseudocode of MxV for CSR-stored matrices



```
for $i \leftarrow 0$ to $max\_nz - 1$ do
    for $j \leftarrow jd\_ptr[i]$ to $jd\_ptr[i+1] - 1$ do
        $r \leftarrow j - jd\_ptr[i]$
        $y[perm[r]] \leftarrow y[perm[r]] + values[j] \times x[col\_ind[j]]$
```

Fig. 2. A JDS example and pseudocode of MxV for JDS-stored matrices

matrix rows are reordered according to the number of nonzeros in each row in decreasing order. Then, all nonzeros of the matrix are shifted to the left. Columns of the new compressed matrix are called jagged diagonals. Nonzero values of the compressed matrix are stored in an array in column order. Corresponding column indices of each nonzero in the original matrix are written in another array. One more array is kept to point the beginning indices of each jagged diagonal. Finally row permutation is stored in an array, where elements of the array that correspond to the rows of the compressed matrix, point the row number in the original matrix. Fig. 2 depicts an example for JDS and the pseudocode for MxV of JDS-stored matrices. In the figure, perm, jd_ptr and col_ind respectively stand for permutation, jagged diagonal pointer and column index arrays. max_nz is the maximum of the number of nonzeros of each row.

2.2 Mixed Precision Iterative Refinement for Conjugate Gradients

Conjugate Gradient method is used to solve linear systems $\mathbf{Ax} = \mathbf{b}$, where matrix $\mathbf{A}$ is symmetric and positive definite. Solvers on GPUs that do not support double precision floating point operations suffer from loss of accuracy in the result. Therefore, in this work we adopt a mixed precision iterative refinement algorithm for CG [6] which is based on inner-outer iteration method [9]. The algorithm explained in [6] is tested on conventional processors, but reported to be applicable on GPUs, also. Authors report that the mixed precision algorithm achieves faster solution of the same or even better accuracy compared to the full double precision solver.

Basically, mixed precision algorithm runs the preconditioned CG. However, instead of using a fixed preconditioner, preconditioner is solved using a single precision sparse iterative method, in each iteration. Operations other than the inner solver run in double precision. Single precision inner solver may also use preconditioned CG method if a preconditioner is available or any other iterative method that result in symmetric and positive definite operations. Inner solver runs for a predetermined number of iterations and takes most of the time of the overall solution.

2.3 General Purpose Computation on GPUs and CUDA

New generation GPUs can be thought as many-core stream-processing units. Taking into account the superiority of peak performance over conventional CPUs, GPUs are great resources not only for graphics processing, but also for data-parallel computing. Using GPUs for non-graphics applications is not a new idea, but with development of new APIs that hide the graphics-related interface and drastic increase in hardware performance, usability and popularity of general purpose computing on GPUs increased significantly [7].

Compute Unified Device Architecture (CUDA) is NVIDIAs new generation GPU architecture. It is also the name of the software for programming this architecture. A CUDA GPU contains number of SIMD multiprocessors. GPU has a device memory that is accessible by all processors. Each multiprocessor contains its own shared memory and read-only constant and texture caches that are accessible by all processors within the multiprocessor. CUDA API supports programming different memory types.

CUDA GPU devices are capable of running high number of threads in parallel. Threads are grouped together as thread blocks, so that each block of threads are executed on the same multiprocessor. As a result, threads in the same block can communicate through fast shared memory.

Threads in different blocks can communicate through device memory. However, access to the device memory is very slow compared to the shared memory. Hence, device memory accesses should be refrained as possible and accesses should be coalesced to attain high performance. If memory access is organized in the right pattern, half of the threads that are scheduled to execute instructions in the same time and in the same block can access to the device memory in a single coalesced read or write instead of many simultaneous accesses. Coalescing is possible if threads access consecutive memory addresses of 4, 8 or 16 bytes and base address for coalesced access should be multiple of 16 times size of the memory type accessed by each thread.

CUDA supports single precision floating point operations based on IEEE 754 standard, with some deviations [2].

2.4 Sparse Iterative Solvers on GPU

GPU memory can be efficiently utilized for solvers where the matrix has a regular structure [10,11]. In this work, our target is to solve systems with irregular sparsity.

The first GPU-based Conjugate Gradient solver for unstructured matrices is proposed by Bolz et al. [4]. To utilize memory bandwidth, blocked CSR (BCSR) matrix storage is used in [5] instead of CSR. BCSR decreases number of memory fetches from the device memory to some extent, however number of elements to be multiplied increases. They achieve 1 to 6.5 Gflops CG performance on QuadroFX 5600 card with a limited dataset of 5 matrices.

Both of the above-mentioned works solve systems in single precision floating point. Göddeke et al. propose mixed precision solutions for FEM simulations on banded matrices [10]. They extend the multi-grid solver to run on a GPU-enhanced cluster in [11]. Georgescu and Okuda [8] use an iterative refinement

```
$mult \leftarrow 0$
for $i \leftarrow 0$ to $nz\_count[t] - 1$ do
    $j \leftarrow jd\_ptr[i] + t$
    $mult \leftarrow mult + values[j] \times x[col\_ind[j]]$
$y[perm[t]] \leftarrow mult$
```

Fig. 3. CSR-like multiplication of JDS-stored matrix. Code for GPU thread t.

algorithm [14] to obtain double precision accuracy, for general matrices. They do not make considerable afford for faster kernel operations, instead prefer a naïve implementation based on CSR format.

3 GPU-Enhanced Conjugate Gradient Solver

We implement the mixed precision algorithm explained in [6] and summarized in Section 2.2. Core operations of single precision inner solver run on the GPU, while double precision refinement iterations run on the CPU. We implement CG for inner solver, assuming that we have no preconditioner readily available.

CG consists of several kernel operations: MxV, SAXPY, vector dot product, norm and scalar operations. Since MxV dominates the running time, fast implementation of MxV is required for faster CG.

3.1 Efficient Sparse Matrix-Vector Multiplies on GPU

We propose an efficient MxV algorithm on GPUs based on JDS storage and CSR-like multiplication. The matrix is stored in JDS format to achieve coalesced reads from the device memory. Each GPU thread multiplies one row of the matrix and computes one output vector element. The proposed MxV procedure for each GPU thread t is depicted in Fig. 3. Note that, for each thread to realize multiplication we need to have one more array (called nz_count in the figure) to store number of nonzeros of each row.

In this multiplication scheme, consecutive threads access to the consecutive indices in arrays values, col_ind, nz_count and perm. So, reads from these arrays can be coalesced, instead of many simultaneous reads. Note that, as mentioned in Section 2.3, to achieve coalescing base addresses of coalesced reads should be multiple of 16 times size of the data type to be read. Namely, elements of jd_ptr should be multiple of 16. We pad zeros to arrays values and col_ind for each jagged diagonal to have multiple of 16 entries.

Unfortunately, writing to the output vector y cannot be coalesced because of the irregular access caused by perm array. Still, unlike JDS multiplication given in Fig. 2, CSR-like multiplication has an advantage of writing the output vector only once.

Since in JDS format matrix rows are sorted according to their nonzero count in decreasing order, better computational load balance is naturally obtained. The variation of nonzero counts between threads within the same block is smallest.

In JDS format, row indices are not consecutively ordered. To access values in row t, indices should be calculated using jd_ptr array. Many accesses to this array may be costly if it is deployed to the device memory. We place this array to read-only constant cache to avoid slow reads. Reading from constant cache is as fast as reading from registers, if active threads in the same block read the same address. In case of cache miss, cost of reading from constant memory is equal to reading from device memory. Size of the cached array jd_ptr is equal to the maximum of the number of nonzeros of rows. For all of our experimental data, this array completely fits into constant cache, hence no cache miss occurs.

We bind x array to the texture cache in our implementation.

3.2 Other Operations

Not only MxV, but all operations of the inner CG solver other than scalar division operation are efficiently implemented on the GPU. Dot products and norms are implemented as in the parallel reduction example of NVIDIA's CUDA SDK [12], in log$n$ steps, where n is the size of the vectors in computations. Each output element of SAXPY operation is calculated by a different thread.

3.3 Multi-GPU Algorithm

CUDA supports multiple GPUs run together for an application. Compared to main memory, GPUs have limited device memory. For applications which require a lot of storage, device memory limitations may be a bottleneck for GPUs. For this reason, sometimes running algorithms on multiple GPUs is not only required for faster applications but to overcome memory bottleneck.

We propose a data-parallel CG algorithm to run on multiple GPUs and a CPU located on the same board. Rows of the matrix and corresponding vector entries are distributed amongst GPUs. Since MxV takes most of the iteration time, we assign nonzeros of the matrix equally to each GPU, so that loads of MxV is balanced amongst GPUs.

CPU creates a thread for each GPU and coordinates the communication amongst them. In every iteration, each GPU communicates with the CPU for them to exchange input vector entries of the matrix-vector multiply. Host CPU holds a global array, where each GPU writes to and reads from for communicating vector entries. When inner solver terminates, solution vector computed by the inner solver is copied to the CPU and refinement iteration on the CPU begins.

Other than above-mentioned communications, scalars are communicated between GPUs and the CPU to compute global results of the locally computed values. In the resulting algorithm, threads synchronize in three points: once before MxV to exchange input vector of the multiplication, and two times to compute scalars.

4 Experimental Results

We evaluate performance of proposed MxV and CG methods on single and multiple GPUs. In our experiments, an AMD Phenom 9850 2.5 GHz Quad-Core

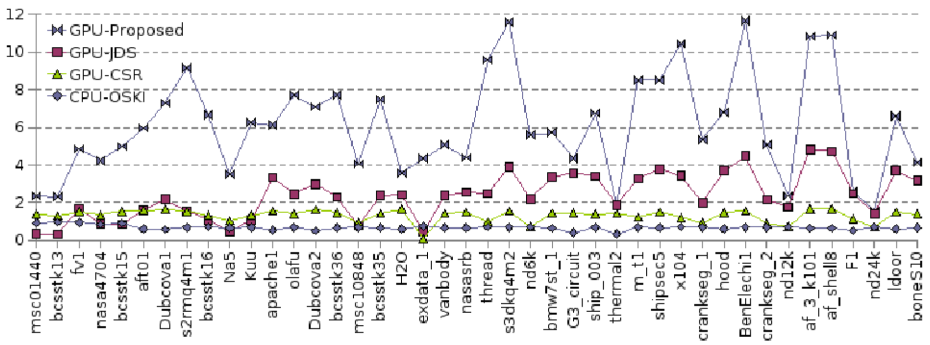


Fig. 4. MxV performance comparison of the proposed algorithm

processor, 4 GB main memory and four GeForce 8800 GTS 512 GPU cards are used in the hardware platform. CUDA version 2.1 is used for coding on Linux 2.6.23 OS.

42 matrices that are symmetric and positive definite with real value entries from Sparse Matrix Collection of University of Florida [3] are used for performance evaluation. Matrix dimensions vary from 1,440 to 1,585,478 and number of nonzeros vary from 46,270 to 55,468,422.

4.1 MxV Performance

We compare performance of the proposed algorithm on the GPU with a CPU implementation using highly optimized OSKI sparse matrix kernel library [18]. To demonstrate the validity of the algorithmic improvements, we also compare our algorithm with basic JDS and CSR implementations on the GPU. Each thread block contains 512 threads for all implementation on the GPU. In the CSR implementation, each GPU thread multiplies one row of the matrix. For JDS implementation, each thread computes one element of the matrix and we pad zeros to matrices to achieve coalescing, as explained in Section 3.1. Since output vector y is accessed number of nonzero times for basic JDS multiplication, accesses on this array in irregular order drastically degrades the performance. Hence, we let each thread t multiplying a nonzero in row r to write r^{th} index of a temporary array y_temp. By this way, writes on y_temp can be coalesced. In the end of the multiplication, y_temp is reordered into y, using the row permutation array.

Comparison results are given in Gflops in Fig. 4. Matrices on x axis are sorted according to the number of nonzeros they contain. GPU-Proposed stands for our proposed multiplication algorithm based on JDS storage and row multiplication. GPU-CSR and GPU-JDS stand for basic CSR and JDS implementations on GPU, respectively. CPU-OSKI stands for our CPU implementation using OSKI.

During MxV, for each nonzero, one multiplication and one addition operation is executed. Therefore, we calculate flops by dividing two times number of nonzeros by execution time. CPU, CSR-based GPU, JDS-based GPU and our algorithm respectively achieve 0.68, 1.32, 2.38 and 6.09 Gflops, on average for matrices in the dataset.

Variation of the performance of our algorithm and JDS-based implementation on test matrices is greater than other implementations. Utilization of GPU resources is lower for smaller matrices, hence the algorithm is slower. This is more obvious for JDS-based implementation. Matrix structure greatly affects our algorithm's performance. For sparser matrices such as G3_circuit and thermal2 (4.8 and 7 nonzeros per row, respectively), and denser matrices such as exdata_1 and nd24k (378.2 and 398.8 nonzeros per row, respectively), algorithm is slower than the average. The algorithm is also slower for matrices whose nonzeros are distributed over the whole matrix, such as F1. This is due to the large number of cache misses on the input vector of multiplication. On the other hand, algorithm is faster for matrices whose consecutive rows (columns) share many columns (rows). For instance, s3dkq4m2, whose nonzeros are ordered around the diagonal, is one of the fastest in our dataset.

Note that the main reason behind the outstanding performance of our algorithm is memory coalescing. In our experiments, we found out that CSR implementation on GPU performs slightly better than our algorithm, if we do not pad 0s to jagged diagonals hence coalesced memory reads do not occur.

Effective memory bandwidth of the proposed MxV algorithm is 36.3 GB/s for matrices in the dataset, on average. The GPU used in experiments has maximum memory bandwidth of 64 GB/s, that is, bandwidth utilization of our algorithm is 57%.

4.2 CG Performance

In this section, we evaluate the performance of single and multiple GPU CG algorithms and CG on the CPU. Performance of inner CG solver iterations of the mixed precision algorithm is given in flops. We do not make convergence analysis of the mixed precision algorithm, since exclusive analysis is already done in [6] and the algorithm is shown to be faster than double precision CG on conventional CPUs while not sacrificing accuracy of the solution. Also, we do not observe any increase in number of iterations for GPU-enhanced algorithms to achieve same accuracy with CPU implementation of both inner and outer solver. Since double precision operations occupy only a small portion of the overall execution time, the performance of the mixed precision algorithm increases speeding up the single precision inner solver. We implement double precision outer iterations on the CPU.

Performance of the inner CG solver on different platforms is given in Fig. 5. The chart on the left depicts CG performance in Gflops and the chart on the right depicts the speedups of the multi-GPU algorithm over single GPU algorithm. For smaller matrices in the dataset, communication cost between CPU and GPU dominates the overall execution time of multi-GPU algorithm, hence employing more GPUs do not increase performance of the solver. For this reason, in the figure, we omit results for matrices that have less than 5 million nonzeros. CG performance is 5.01, 8.62 and 13.85 Gflops with single GPU, 2 GPUs and 4

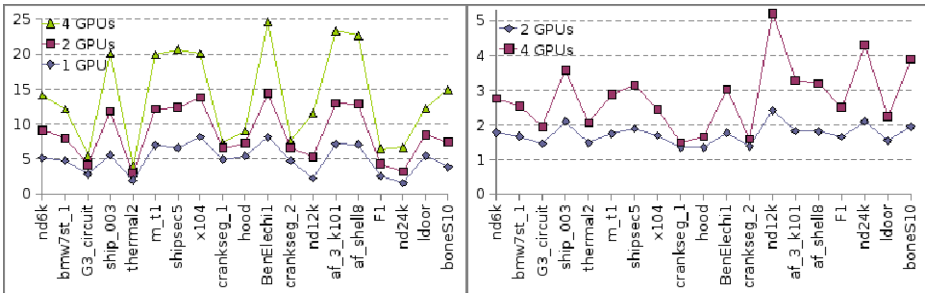


Fig. 5. CG (inner solver) performance. Left: comparison of single and multi-GPU algorithms with CPU implementation Right: Multi-GPU speedups over single GPU .

GPUs, respectively, on average. This means that 2 GPUs speeds up single GPU algorithm by 1.73 and 4 GPUs by 2.83.

It is interesting to observe the big variation on the speedups. We observe only slight speedups for some matrices, while for two matrices we observe superlinear speedups. The most important factor affecting the performance of the multi-GPU algorithm is number of communicating input vector entries between the CPU and GPUs. Some of the other factors affecting the performance are the balance of vector sizes between processors and the cache affect.

For the densest matrix in the dataset, nd24k, we observe superlinear speedups, while for the sparsest matrix, G3_circuit, speedup is only 2 for 4 GPUs. The imbalance of vector sizes of nd24k on 4 GPUs is 24%. During 4-GPU CG computation, while communicating input vector entries for MxV, maximum sending GPU sends 86 KB to the CPU and maximum receiving GPU receives 166 KB from the CPU. Since MxV dominates the total CG time, imbalance of vector sizes become negligible. On the other hand, for the sparsest matrix G3_circuit imbalance of vector sizes is 6%, maximum sending GPU sends 730 KB and maximum receiving GPU receives 732 KB, for 4 GPUs. Here, GPU communication dramatically affects the performance, since computation count per communicating data is very low. For crankseg_2, where speedup is below 2 for this matrix on 4 GPUs, the imbalance of vector sizes is 35%, maximum sending GPU sends 86 KB and maximum receiving GPU receives 207 KB receives almost all vector entries that it does not compute. The density and number of nonzeros of this matrix is about half of nd24k. Still, we cannot explain the performance difference between these two matrices by just the communication and vector imbalance. The speed down of the other crankseg matrix implies that matrix structure also affects the performance difference among matrices. We found out that the variation of number of jagged diagonals across matrices assigned to different GPUs is incomparably high for crankseg matrices, where for nd24k, there is almost no variation. Sparsity patterns of distributed matrices across GPUs are too different for crankseg matrices, so that even we assume no communication, ideal speedup is impossible due to the difference in cache utilization of parallel GPUs. For these types of matrices, a clever row distribution algorithm emerges.

5 Conclusion and Future Work

In this work, we have demonstrated efficient utilization of GPUs for solution of general sparse symmetric linear systems with double precision solution accuracy. We have implemented a mixed precision CG solver on multiple GPUs and evaluated its performance on a wide range of matrices. The performance of the proposed algorithm reveals that stream processing on modern GPUs can be useful for increasing memory bandwidth utilization for sparse linear solvers. The proposed MxV algorithm, which is the most time-consuming operation of the CG solver, utilizes constant and texture caches, as well as coalesced memory reads from the device memory. As a result, we achieve the fastest MxV implementation for unstructured sparse matrices on the GPU, to the best of our knowledge.

Number of cache misses on input vector considerably affects the run time of MxV. It is very difficult to find algorithms to increase the cache utilization for input vector. There are some works dedicated to decrease the number of cache misses for CSR multiplication, which is an NP-complete problem [15,16]. We plan to study on better cache utilization for the input vector of the proposed MxV algorithm in future.

Our multi-GPU algorithm achieves 13.85 Gflops for CG on 4 GPUs, on average. Communication between GPUs and the CPU significantly degrades the performance of the multi-GPU algorithm. Performance of the multi-GPU algorithm can be further increased by direct GPU to GPU communication. We await CUDA support for direct inter-GPU communication instead of communication through the CPU.

In the future, we plan to study on scalable implementation of the parallel algorithm to run on a GPU cluster. Although we demonstrate results only for the CG solver in this work, proposed techniques can also be applicable for other symmetric or asymmetric iterative solvers.

References

1. ATI CTM Guide Technical Reference Manual. Advanced Micro Devices, Inc. (2006)
2. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. NVIDIA Corporation (2007)
3. University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/
4. Bolz, J., Farmer, I., Grinspun, E., Schröder, P.: Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. ACM Transactions on Graphics (2003)
5. Buatois, L., Caumon, G., Lévy, B.: Concurrent Number Cruncher: A GPU Implementation of a General Sparse Linear Solver. International Journal of Parallel, Emergent and Distributed Systems (to appear)
6. Buttari, A., Dongarra, J., Kurzak, J., Luszczek, P., Tomov, S.: Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy. ACM Transactions on Mathematical Software (2008)
7. Blythe, D.: Rise of the Graphics Processor. Proc. of the IEEE 96(5) (2008)
8. Georgescu, S., Okuda, H.: GPGPU-Enhanced Conjugate Gradient Solver for Finite Element Matrices. In: Proc. iWAPT, Tokyo (2007)

9. Golub, G.H., Ye, Q.: Inexact Preconditioned Conjugate Gradient Method with Inner-Outer Iterations. SIAM J. on Scientific Computing 21(4), 1305–1320 (2000)
10. Göddeke, D., Strzodka, R., Türek, S.: Accelerating Double Precision FEM Simulations with GPUs. In: Proc. ASIM 2005 (2005)
11. Göddeke, D., Strzodka, R., Mohd-Yusof, J., McCormick, P., Wobker, H., Becker, C., Türek, S.: Using GPUs to Improve Multigrid Solver Performance on a Cluster. International Journal of Computational Science and Engineering (2008)
12. Harris, M.: Optimizing Parallel Reduction in CUDA. NVIDIA Dev. Tech. (2007)
13. Krüger, J., Westermann, R.: Linear Algebra Operators for GPU implementation of Numerical Algorithms. ACM Transaction on Graphics 22(3), 908–916 (2003)
14. Martin, R.S., Peters, G., Wilkinson, J.H.: Iterative Refinement of the Solution of a Positive Definite System of Equations. Numerische Mathematik 8, 203–216 (1966)
15. Pichel, J.C., Heras, D.B., Cabaleiro, J.C., Rivera, F.F.: Improving the Locality of the Sparse Matrix-Vector Product on Shared Memory Multiprocessors. In: Proc. PDP 2004 (2004)
16. Pinar, A., Heath, M.T.: Improving the Performance of Sparse Matrix-Vector Multiplication. In: Proc. SC 1999 (1999)
17. Saad, Y.: SPARSEKIT: A Basic Tool for Sparse Matrix Computation. Tech. Rep. CSRD TR 1029, Univ. of Illionis, Urbana, IL (1990)
18. Vuduc, R., Demmel, J., Yelick, K.: OSKI: A Library of Automatically Tuned Sparse Matrix Kernels. In: Proc. SciDAC 2005 (2005)

CUDA Solutions for the SSSP Problem[*]

Pedro J. Martín, Roberto Torres, and Antonio Gavilanes

Dpto. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain
{pjmartin@sip,r.torres@fdi,agav@sip}.ucm.es

Abstract. We present several algorithms that solve the single-source shortest-path problem using CUDA. We have run them on a database, composed of hundreds of large graphs represented by adjacency lists and adjacency matrices, achieving high speedups regarding a CPU implementation based on Fibonacci heaps. Concerning correctness, we outline why our solutions work, and show that a previous approach [10] is incorrect.

Keywords: Shortest path algorithms, GPU, CUDA.

1 Introduction

Computing shortest paths in a graph is one of the most fundamental problems in computer science and network optimization. In particular, the *Single-Source Shortest-Paths* (in the sequel, SSSP) problem, which computes the weight of the shortest path from a specific vertex (source) to all other vertices, in a weighted directed graph, is a heavily studied problem in graph theory.

Probably, the most well-known algorithm solving this problem for the case of graphs with nonnegative edges was given by Dijkstra in 1959 [1], and nearly all the subsequent proposals are based on it. In spite of its early formulation, this classic solution is still presented in almost every textbook on algorithms [2]. After the simplest Dijkstra´s implementation, which uses arrays to represent min-priority queues and runs in $O(n^2)$ time, where n is the number of vertices, many authors have designed different data structures to implement these queues and achieve better and better asymptotic running times. In particular, Fibonacci heaps [3] can be used to get $O(m + n \log n)$, where m is the number of edges.

As [4] points out, Dijkstra's algorithm is inherently sequential since its efficiency depends on a fixed ordering of the vertices. The Bellman-Ford algorithm allows all vertices to be considered in parallel but at the cost of being not efficient. Different formulations of parallel algorithms for the SSSP problem are reviewed in detail in [5]. In particular, a specific proposal for incorporating parallelism into Dijkstra's algorithm has been the introduction of parallel priority queues [6]. However, the literature contains few experimental studies on parallel algorithms of the nonnegative SSSP problem. Some of the more recent works study the use of supercomputers for solving large graphs. [7] reports performance results on the multithread parallel computer Cray MTA-2, using the Δ-stepping parallel algorithm of [5]. [7] exhibits remarkable parallel speedup when compared to competitive sequential algorithms, for

[*] Research supported by the Project CCG08-UCM/TIC-4252.

low-diameter sparse graphs of 100 million vertices and 1 billion edges. On the other hand, [8] contains an experimental evaluation of [6] on the APEmille supercomputer, but restricted to graphs with no more than thousands of vertices.

However, some modern applications, such as data mining, network organization, etc. require large graphs with millions of vertices, and some of the previous algorithms become impractical, when we do not have a very expensive hardware at our disposal. Fortunately, Graphics Processing Units (GPUs) supply a high parallel computation power at a low price. Moreover, they have become very popular since the languages involved in their programming have evolved from graphics APIs to general purpose languages. One of the best examples is the CUDA API [9] of NVIDIA. As a consequence of this evolution, the so called General Purpose Computing on GPU (GPGPU) [11] has consolidated as a very active research area, where many problems that are not directly related to computer graphics are solved using GPUs. The aim of all these GPU-based implementations is to achieve better running times than their CPU-based counterparts.

With this new technology available, the natural challenge is: "how can GPUs be used to solve the SSSP problem?". Unfortunately programming with CUDA must be carefully taken, basically because CUDA programming model is very restricted concerning synchronization, and the unique proposal we are aware of ([10]) is not correct. Apart from giving a counterexample, in this paper we present different correct solutions, based on Dijkstra´s algorithm, that are experimentally compared using a database of hundreds of randomly generated large graphs.

2 Dijkstra´s Algorithm Overview

Dijkstra´s algorithm solves the SSSP problem for directed graphs $G = (V, E)$ in which every edge $(v, v') \in E$ has a positive weight $\omega(v, v') > 0$. Let n and m be the number or vertices and edges respectively. We assume that vertices are numbered from 0 to $n - 1$, and that 0 is the source vertex. The algorithm splits the set of vertices in two parts: the set R of *resolved vertices* (R-vertices) and the set U of *unresolved vertices* (U-vertices), and it keeps a shortest-path estimate $c[i]$ for each vertex i, which actually coincides with the shortest path weight for R-vertices. For U-vertices, $c[i]$ holds the weight of the *shortest special path* (SSP) to i w.r.t. R, that is, the shortest path among the paths to i that exclusively traverses R-vertices before reaching i.

The algorithm implements a loop. Each iteration is composed of three steps: (1) the estimates for U-vertices are relaxed using the last vertex added to R, which we will call the *frontier vertex*, (2) the minimum estimate for U-vertices is computed, and (3) a U-vertex with the minimum estimate is promoted to R, and becomes the new frontier. Figure 1 presents a typical Dijkstra's algorithm implementation that includes the variable `f` to hold the current frontier vertex. Regardless of the graph representation we chose, it runs in $O(n^2)$.

The soundness of the algorithm is based on two fundamental properties that can be proved. First, anytime a new frontier arises in the third step, its estimate actually coincides to the weight of its shortest path, thus it can be safely promoted to R. Second, in order to relax the estimates of a U-vertex j using the current frontier vertex f, the SSP to j w.r.t. R cannot traverse more R-vertices after visiting f, hence we only consider the previous estimate and $c[f] + \omega(f, j)$ when updating $c[j]$.

```
void Dijkstra (c) {
  forall vertex i { c[i]=INFINITY; u[i]=true;}
  c[0]=0; u[0]=false;
  f=0; mssp=0;
  while (mssp!=INFINITY) {
    forall unresolved vertex j {
         c[j]= min(c[j], c[f]+w[f,j]);}
    mssp= INFINITY;
    forall unresolved vertex j
        if(c[j]<mssp){ mssp=c[j]; f=j;}
    u[f]=false;
  }//while
}
```

Fig. 1. Dijkstra's algorithm implementation

3 Parallelizing Dijkstra's Algorithm

Dijkstra´s algorithm handles a unique frontier vertex even when the estimates of several U-vertices coincide with the minimum computed in the second step. In these cases, the algorithm simply chooses one of them to compose the new frontier. In consequence, it requires a different iteration to promote each of them to R. Fortunately, this set of U-vertices, which we will call F, can be processed at once because the previous two properties remain:

1. Their estimates actually coincide with the weight of their shortest paths.
2. In order to relax the estimate of a remaining U-vertex j, the SSP to j w.r.t. R cannot traverse more R-vertices after visiting one F-vertex, hence only the previous estimate and $\min_{f\in F}\{c[f]+\omega(f,j)\}$ must be considered when updating $c[j]$. In particular, note that only one F-vertex can belong to the SSP to j w.r.t. R.

```
void DA2CF(c) {
  initialize(c, f, u);
  mssp = 0;
  while (mssp != INFINITY) {
    relax(c, f, u);
    mssp = minimum(c, u);
    update(c, f, u, mssp);
  }//while
}
```

```
void initialize(c, f, u) {
  forall vertex i {
    c[i] = INFINITY;
    f[i] = false;
    u[i] = true;
  }//for
  c[0] = 0;
  f[0] = true;  u[0] = false;
}
```

Fig. 2. Dijkstra's algorithm adapted to compound frontiers

Therefore the notion of *compound frontier* can be used to design the *Dijkstra's algorithm Adapted to Compound Frontiers* (DA2CF) presented in Fig. 2. Although the algorithm is composed of the same three basic operations, their implementations must suitably handle compound frontiers:

1. `relax(c, f, u)` must relax the shortest path estimate for every U-vertex using F-vertices. Hence it must compute $c[j] = min\{c[j], c[f] + \omega(f,j)\}$ for every pair of vertices $j \in U$ and $f \in F$.
2. `minimum(c, u)` must find the minimum estimate of the U-vertices, called `mssp`.
3. `update(c, f, u, mssp)` must update the set of U-vertices by removing those vertices whose estimate is equal to `mssp`, which will compose the new set of F-vertices.

There are many ways to implement these operations. Although sequential solutions could be easily written by means of the obvious single loop (two nested loops for the `relax` procedure), the operations can be performed in parallel, by launching a thread for each iteration of the loop (the main loop for `relax`).

Figure 3 shows two versions of the `relax` procedure. On the left, `relax_F` processes F-vertices: "for each F-vertex we visit all of its successors, relaxing c for those vertices that are still unresolved". Observe that the sentence `c[j]=min(c[j], c[i]+ω(i,j))` could produce concurrency inconsistencies if two F-vertices `i` and `i'` accessed the same U-vertex `j` and the worst value `c[i]+ω(i,j)` were finally left. In order to prevent such inconsistencies, we use the atomic instruction `atomicMin(x,y)` that allows only one thread to store the minimum of `x` and `y` in the variable `x`.

CUDA devices of compute capability 1.0 do not support atomic functions, thus we propose another approach that does not use them. Figure 3 on the right presents the `relax_U` procedure which focuses on U-vertices instead of F-vertices: "for each U-vertex, we visit all of its predecessors, relaxing its c-value when a F-vertex is found". Notice that this approach requires predecessors instead of successors.

A parallel version of the `minimum` function is a more difficult task, because of its sequential nature. Fortunately, different reduction procedures have been already adapted to the stream model [12, 13, 14]. In this paper we have adapted the `reduce3` method included in the CUDA SDK 1.1 [15] to obtain the `minimum1` procedure of Fig. 4 on the right.

Finally, we parallelize the `update` procedure as Fig. 4 on the left shows. In the sequel, DA2CF_F and DA2CF_U will denote the algorithms that use `relax_F` and `relax_U`, respectively.

Regarding asymptotic complexity, let us compare the Dijkstra´s algorithm of Fig. 1, that runs in $O(n^2)$, to the sequential versions of the DA2CF algorithm that result

```
void relax_F(c, f, u) {
  forall i in parallel do {
    if (f[i]) {
      forall j successor of i do {
        if (u[j])
          atomicMin(c[j],c[i]+w[i,j]);
      }//for
    }//if
  }//for
}
```

```
void relax_U(c, f, u) {
  forall i in parallel do {
    if (u[i]) {
      forall j predecessor of i do {
        if (f[j])
          c[i]= min(c[i],c[j]+w[j,i]);
      }//for
    }//if
  }//for
}
```

Fig. 3. Processing frontier (left) or unresolved (right) vertices within the `relax` operation

```
void update(c, f, u, mssp) {
  forall i in parallel do {
    f[i] = false;
    if (c[i] == mssp) {
      u[i] = false;
      f[i] = true;
    }//if
  }//for
}
```

```
void minimum1(u, c, minimums) {
  forall i in parallel do {
    thid = threadIdx.x;
    i = blockIdx.x*(2*blockDim.x)+threadIdx.x;
    j = i + blockDim.x;
    data1 = u[i] ? c[i] : INFINITY;
    data2 = u[j] ? c[j] : INFINITY;
    sdata[thid] = min(data1, data2);
    __syncthreads();
    for (s = blockDim.x/2; s>0; s>>=1) {
      if (thid<s) {
        sdata[thid]=min(sdata[thid],sdata[thid+s]);
      }// if
      __syncthreads();
    }// for
    if (thid==0) minimums[blockIdx.x]= sdata[0];
  }// forall
}
```

Fig. 4. Updating the frontier (left), and computing the minimum sssp with CUDA (right)

when the "in parallel" qualifier is erased. Firstly, notice that the number of iterations required for the main DA2CF-loop depends more heavily on the given graph; since the size of the arising compound frontiers influences its termination. Hence, we analyze its worst case. We focus on adjacency lists since they fit better to large graphs and they provide the algorithm of Fig. 1 with smaller running times. The worst case corresponds to a complete graph requiring n iterations (the frontier size is always 1), DA2CF_F also takes a time in $O(n^2)$, but with a greater constant due to the management of the `f` array. However, DA2CF_U takes a time in $O(n^3)$, since the edges arriving at an unresolved vertex are repeatedly processed while it remains unresolved. In order to evaluate the general case, we experimentally run CUDA implementations on randomly generated graphs.

4 CUDA Implementations

The adjacency list representation of a graph is made up of three arrays: v for vertices, e for edges and ω for weights. Array v is used to access the adjacency list of each vertex. Specifically, the adjacency list of the vertex i appears in e and ω from index $v[i]$ to index $v[i+1]-1$ (Fig. 5 on the left). In order to deal with the last vertex in the same way, an extra component is added at the end of v such that $v[n]=m$. In consequence, array v is of size $n+1$ and both e and ω are of size m.

There are two possible interpretations for the data occurring in e. Vertices belonging to the adjacency list of vertex i can be understood as successors or predecessors. Formally, in the predecessor interpretation, there is an edge to i from each adjacent vertex, whereas in the successor interpretation the edge goes from i to each adjacent vertex. Graphs must be represented in the proper interpretation before execution, since the `relax` procedure requires either successors (`relax_F`) or predecessors (`relax_U`), but not both.

4.1 Implementations

We have sequential C implementations corresponding to the sequential versions of DA2CF_F and DA2CF_U, that we respectively call F^{CPU} and U^{CPU}. Before presenting the pure CUDA implementations, we have tried some hybrid systems running on both, CPU and GPU. Since the `minimum` function is inherently sequential, we have restricted this function to run on CPU. Moreover, in order to fit the requirements of any CUDA device, we have focused on the `relax_U` procedure. Hence, we have designed three hybrid implementations based on the DA2CF_U algorithm: U^{H1}, U^{H2} and U^{H3} which respectively run the `update` procedure, the `relax_U` procedure, and both `update` and `relax_U` on the GPU.

In order to run the complete algorithm on GPU, we must run additional passes of the `minimum` function, since the `minimum1` kernel of Fig. 4 only reduces each block to a single value. Thus, we have implemented another kernel, called `minimum2`, to execute a second pass on GPU. The obtained values are finally minimized on CPU in a sequential manner, because the number of these values is too small. Hence, we have two fully GPU-implemented solutions based on the DA2CF_U algorithm, U^{GPU} and $U^{GPU+2min}$ that apply one and two minimization passes, respectively. Based on the DA2CF_F algorithm, we also have two fully-GPU solutions, but this time they have been designed to analyze the cost due to simultaneous accesses to the `c` array. Thus, apart from the F^{GPU} solution, we have another one, called $F^{GPU\_no\_Atomic}$, that does not use the atomic function `atomicMin` but a non-atomic function `min`. We introduce the latter solution only for measuring purposes, since it is not correct in a parallel environment. Anyway, both solutions apply a single `minimum` pass.

4.2 Exploiting CUDA Resources

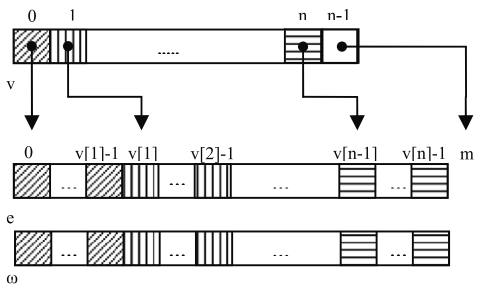
It is possible to accelerate the U^{GPU} solution by using some CUDA features. Concretely, in this subsection we exploit texture cache and shared memory to improve the implementation of the `relax_U` kernel. Let us call the corresponding solution U^{GPU_PLUS}.

In order to retrieve the boundaries of the adjacency list, the i–th thread must access $v[i]$ and $v[i+1]$, whereas the $(i+1)$–th thread must access $v[i+1]$ and $v[i+2]$. Thus, the value $v[i+1]$ is shared by the two threads, and can be brought only once if shared memory is used. Then, each thread i reads $v[i]$ from global memory, writes it to shared memory, and after that, it reads $v[i+1]$ from shared memory directly. A special case is the last thread of a block, since it will bring both $v[i]$ and $v[i+1]$.

Notice that two threads can access the array `f` for the same vertex `j`. To accelerate the corresponding readings, the array can be accessed through a texture, taking advantage of the texture cache. Thus, it is possible for threads of the same block to read `f[j]` from the cache and not from global memory.

4.3 A Bugged Implementation

Let us explain the problem we have found in the solution presented in [10]. The authors propose an implementation of Dijkstra´s algorithm which relaxes using the frontier, in a similar way to our `relax_F` procedure. However, instead of the atomic



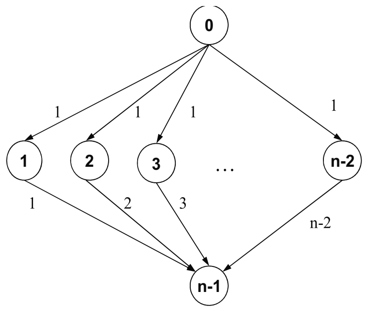


Fig. 5. Left: The adjacency list representation. Right: Counterexample to [10].

function `atomicMin`, they use the following code to relax a vertex `j` which is successor of a frontier vertex `i`:

```
if (uc[j] > c[i]+w[i,j])  uc[j] = c[i]+w[i,j];
```

where the array `uc`, called the *updating cost array*, holds a copy of the array `c` before relaxing. Indeed, as the authors explain, the new cost is not reflected in `c`, but is updated in `uc`, in order to avoid read-after-write inconsistencies. Later, they dump `uc` onto `c` in the `update` kernel.

Unfortunately, this technique is not enough to avoid write-after-write inconsistencies. Concretely, if two frontier vertices `i` and `i'`, whose related threads are simultaneously running, satisfy `uc[j]>c[i]+w[i,j]` and `uc[j]>c[i']+w[i',j]` at the same time for the same U-vertex `j`, then both threads will make the above *if*-condition true. Thus, there will be no control on the final value assigned to `uc[j]`. Since debugging concurrent programs is highly difficult, we have defined the graph of Fig. 5 on the right in order to increase the number of these critical situations.

We have run their implementation on a GeForce 8800 GTS, similar to the GeForce 8800 GTX used in [10], with n=1024 and 32 threads per block. Furthermore, we have also tested the undirected version of the graph, since the authors do not specify the kind of graph they manage. In any case, observe that vertices ranging from 1 to $n-2$ will compose the frontier after the first iteration. Actually, $c[i] = uc[i] = 1$, for $1 \leq i \leq n-2$, and $c[n-1] = uc[n-1] =$ INFINITY, after the first iteration. Hence, every vertex $(1 \leq i \leq n-2)$, tries to relax $uc[n-1]$ to a different value during the second iteration. In consequence $c[n-1]$ ends with a value that randomly changes from execution to execution, instead of computing the right solution $c[n-1] = 2$.

Since threads of different blocks cannot be synchronized in CUDA, solving this bug requires the use of atomic functions in the `relax_F` implementation. Unfortunately such functions are only available from compute capability 1.1, so solving this bug for the cards GeForce 8800 GTS and GTX demands a deeper modification of the algorithm. This is actually the aim of our `relax_U` kernel.

5 Adjacency Matrices

In the case of adjacency lists, it is difficult to conceive a method to allow threads to collaborate when reading from global memory. On the opposite, when adjacency matrices are used, threads must visit every element of each column or row, and so, threads can cooperate to bring elements of arrays `f`, `c` or `u` to shared memory.

As we did for the adjacency list representation, we can consider two kind of implementations: one that looks for predecessors (U^{CPU} and U^{GPU}), and another one that looks for successors (F^{CPU} and F^{GPU}). In U^{GPU}, each thread t must look for its predecessors by visiting the t–th column. In order to make threads collaborate, the exploration is divided in chunks of b elements, where b is the number of threads in a block. The arrays `f` and `c` are also divided in chunks of b elements. Before visiting the chunk of the column, each thread brings a component of the chunk of `f` and `c` into shared memory. That way, the information of the arrays `f` and `c` is already available when each predecessor within the chunk is processed. Once a chunk is dispatched, the next one is processed identically.

On the other hand, F^{GPU} processes frontier vertices, so each thread explores a row. Threads can also collaborate similarly, but this time they bring elements of `u`.

6 Results and Discussion

We have tested all the implementations using an Intel CORE 2 QUAD Q6600 2.40 GHz 2GB DDR memory, and a NVIDIA GEFORCE GTX 280, which has 30 multiprocessors and 1 GB of GDDR3 memory, using 256 threads per block. The database is composed of randomly generated graphs with a number of vertices that ranges from 1 to 11 M for adjacency lists, and from 1 to 15 K for adjacency matrices. The database includes 25 graphs for each of these sizes. The degree of each graph is fixed, so every vertex has the same number of adjacent vertices. The chosen degree is 7 for adjacency lists, while $n/5$ for matrices. Concerning lists, graphs have been generated using the predecessor interpretation, so we have also turned each graph into

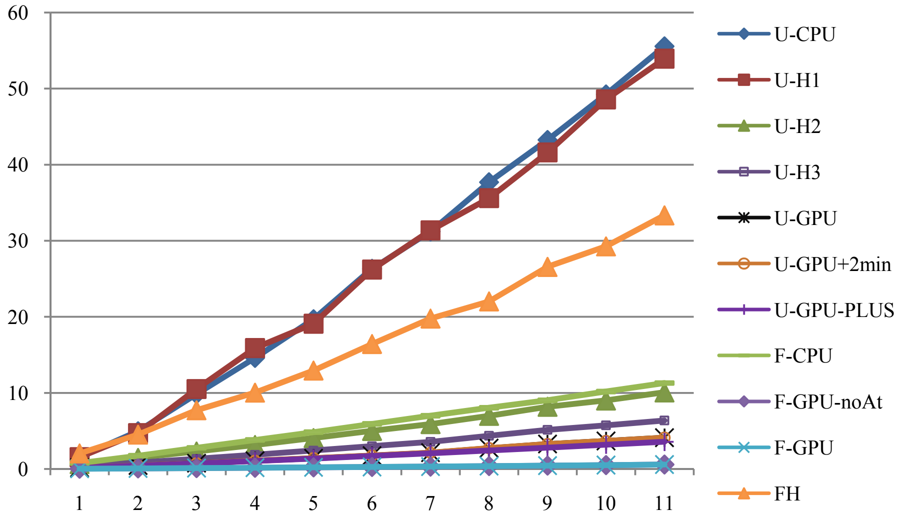


Fig. 6. Results for adjacency lists. Units: seconds for the y-axis and 2^{20} vertices for the x-axis.

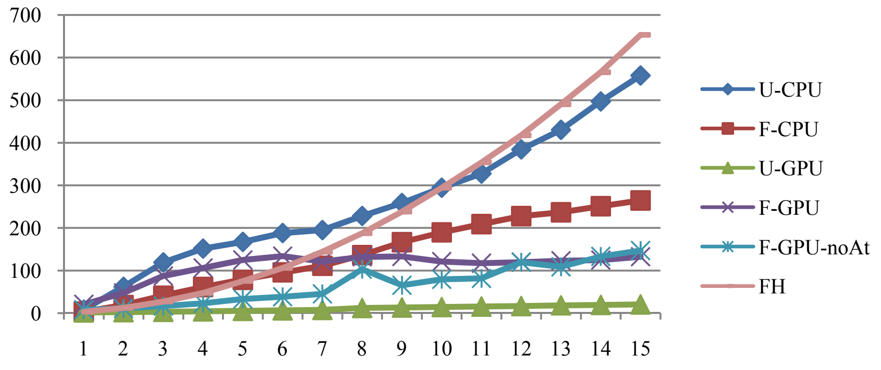


Fig. 7. Results for adjacency matrices. Units: ms. for the y-axis and 2^{10} vertices for the x-axis.

its successor interpretation. Notice that the degree of the graph can not to be kept after this operation. Edge weights are integers that randomly range from 1 to 10.

Figures 6 and 7 show the results we have obtained comparing the average times for each solution to a CPU-solution, called FH, implemented using Fibonacci Heaps and based on the SPLIB library [16]. Most of our solutions, including some CPU ones, run faster on these graphs since the arising frontiers are large. Thus, our solutions only require a few iterations to solve the problem. Concretely, around 45 are enough to solve the largest graphs represented with adjacency lists.

Let us analyze the results for the adjacency list representation presented in Fig. 6. The figure shows that fully CUDA-implemented solutions (U^{GPU}, $U^{GPU+2min}$ and F^{GPU}) are more efficient than partially CUDA-implemented ones (U^{H1}, U^{H2} and U^{H3}), which is due to the overhead connected to the data movement between CPU and GPU. The figure also shows that a two-pass minimization behaves as a single one, since U^{GPU} and $U^{GPU+2min}$ overlap. This can be explained comparing the number of values provided by `minimum1` to those provided by `minimum2`. Notice that these numbers are $n/(2b)$ and $n/(2b)^2$, respectively, where b is the number of threads per block. Since n ranges from $1 * 2^{20}$ to $11 * 2^{20}$ and we have chosen $b = 256 = 2^8$, these numbers finally range from 2^{11} to $11 * 2^{11}$ for `minimum1` and from 2^2 to $11 * 2^2$ for `minimum2`. Therefore, the number of values that must be copied from GPU to CPU, in order to be minimized on CPU, is similar for U^{GPU} and $U^{GPU+2min}$; so there is no difference in time consumption. The figure also shows that exploiting CUDA resources leads to better results, since U^{GPU_PLUS} is slightly faster, obtaining a factor near 10X w.r.t. FH.

Concerning solutions based on the `relax_F` procedure, Fig. 6 shows that processing unresolved vertices is slower than processing the frontier, even for parallel implementations, since F^{GPU} is quite faster than U^{GPU}. Also notice that $F^{GPU\_no\_Atomic}$ behaves as F^{GPU} because the simultaneous accesses to the same `c`-component are rare when the degree is small. These solutions reach a factor around 60X w.r.t. FH.

Regarding adjacency matrices (Fig. 7), the more vertices the graph has, the higher is the degree. Thus, the frontier sets are huge, and `relax_F` based solutions are slower than `relax_U` based ones. To summarize, U^{GPU} is the fastest, achieving a factor of 32X w.r.t. FH. Finally, the figure gives more insight about how atomic operations affect the overall performance, since $F^{GPU\_no\_Atomic}$ is usually faster than F^{GPU}.

7 Conclusions

GPUs can be used to speed up solutions to many problems, including classic problems. Nevertheless, the CUDA programming model is very restricted concerning synchronization, so implementations must be carefully designed, and intuitions about their correctness should be given at least.

In the paper we have shown different CUDA solutions for the SSSP problem, considering adjacency lists and matrices. We have also explained the bug we found in [10], which is basically due to write-after-write inconsistencies. In order to solve this bug, two approaches have been shown. On the one hand, atomic functions can be used for devices of compute capability 1.1 and higher. On the other one, the usual `relax` procedure can be reversed in order to process unresolved vertices instead of frontier vertices. Although processing unresolved vertices is theoretically less efficient, the latter approach is the only applicable solution to any CUDA device.

References

1. Dijkstra, E.W.: A note on two problems in connexion with graphs. Num. Math. 1, 269–271 (1959)
2. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms, 2nd edn. MIT Press, Cambridge (2001)
3. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 596–615 (1987)
4. Meyer, U., Sanders, P.: Δ-stepping: A parallel single source shortest path algorithm. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 393–404. Springer, Heidelberg (1998)
5. Meyer, U., Sanders, P.: Δ-stepping: a parallelizable shortest path algorithm. J. of Algorithms 49, 114–152 (2003)
6. Brodal, G., Träff, J., Zaroliagis, C.: A parallel priority queue with constant time operations. J. Parallel and Distributed Computing 49, 4–21 (1998)
7. Madduri, K., Bader, D., Berry, J., Crobak, J.: An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. In: Proc. Workshop on Algorithm Engineering and Experiments (ALENEX 2007) (2006)
8. Di Stefano, G., Petricola, A., Zaroliagis, C.: On the implementation of parallel shortest path algorithms on a supercomputer. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) ISPA 2006. LNCS, vol. 4330, pp. 406–417. Springer, Heidelberg (2006)
9. `http://www.nvidia.com/object/cuda_home.html#`
10. Harish, P., Narayanan, P.J.: Accelerating large graph algorithms on the GPU using CUDA. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 197–208. Springer, Heidelberg (2007)
11. `http://www.gpgpu.org/`
12. Buck, I., Purcell, T.: A toolkit for computation on GPUs. In: GPU Gems, ch. 37. Addison-Wesley, Reading (2004)
13. Harris, M., Sengupta, S., Owens, J.: Parallel prefix sum (Scan) with CUDA. In: GPU Gems, ch. 39, vol. 3. Addison-Wesley, Reading (2008)
14. Harris, M.: Parallel prefix sum (Scan) with CUDA (2007), `http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/scan/doc/scan.pdf`
15. `http://www.nvidia.com/object/cuda_get.html`
16. `http://avglab.com/andrew/soft.html`

Power Consumption of GPUs from a Software Perspective*

Caroline Collange[1] , David Defour[1], and Arnaud Tisserand[2]

[1] ELIAUS, Univ. de Perpignan, 52 Av. Paul Alduy, 66860 Perpignan, France
{david.defour}@univ-perp.fr

[2] IRISA, CNRS, INRIA, Univ. Rennes 1. 6 rue de Kérampont. BP 80518.
22305 Lannion Cedex, France
arnaud.tisserand@irisa.fr

Abstract. GPUs are now considered as serious challengers for high-performance computing solutions. They have power consumptions up to 300 W. This may lead to power supply and thermal dissipation problems in computing centers. In this article we investigate, using measurements, how and where modern GPUs are using energy during various computations in a CUDA environment.

1 Introduction

As GPUs gained in flexibility through high-level languages such as CUDA, GPUs gained interest for the acceleration of tasks usually performed by a CPU thanks to the high computational power of GPUs. Therefore we are witnessing a tremendous growth in the usage of GPUs for high-performance solutions in computing centers. However, as long as the main goal of GPU was to serve in desktops, their power consumption was secondary. Even though that the ratio of Single Precision GFLOP per W is always in favor of GPUs compared to traditional CPUs (4 SP GFLOP/W for a GTX280 and 0.8 SP GFLOP/W for a core i7 960), these processors are known to have high power consumption. Therefore new challenges need to be solved in order to spread their usage in computing centers where 1 dollar spent in power supply corresponds to 1 dollar spent in cooling system.

This work investigates how and where the power consumption is located within a GPU board by analyzing the relations between the measured power consumption, the required time and the type of units that are stressed to perform a defined operation. There are several solutions to measure or estimate the power consumption of a processor. There exist power estimations based on cycle-level simulation like in Wattch [1] or SimplePower [6] that rely on a low-level description of the architecture which is unavailable for current GPUs. Functional approaches and tools such as SoftExplorer [4] to estimate power consumption at assembly or C levels were proposed for VLIW processors or DSPs. Another

* This work is sponsored by the French ANR (BioWic project) as well as generous hardware donations from nVidia corporation.

solution is to physically measure the power consumption. The measurements can also be used to calibrate some models used for power estimation. Previous work have shown that in the context of multimedia applications for modern out of order processors, CMP is more energy efficient than SMT [3]. [5] and [2] provide good references for power consumption and reduction aspects.

In this paper we consider Nvidia GPUs used for GPGPU (General Purpose computing using GPU) in a CUDA environment. During the analysis, functional blocks are identified, and their consumption is characterized using physical measurements. The considered blocks correspond to units that are usually stressed while executing common kernels on the GPU: register file, memory hierarchy and functional units. In addition to the power estimation, our analysis gives us some information on the organization of the memory hierarchy, the behavior of functional units and some undocumented features.

A modern GPU board is described in Section 2. The measurement process and the parameters extraction method from CUDA are described in Section 3. Results and their analysis for several GPUs are presented in Section 4.

2 Description of a GPU Card

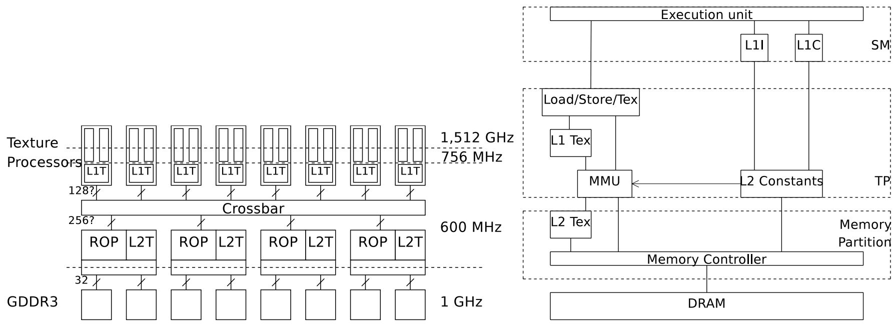
High-end GPUs considered in this work are sold with other component on a dedicated graphics card. These components consist in the GPU, the graphics BIOS, graphics memory, digital to analog converters, dedicated accelerators, an interface with the motherboard, a cooling device and power adapters.

While CPU manufacturers have moved toward higher power efficiency, the power required by GPUs has continued to rise. Although the power provided by power supplies have followed the necessary trend, a PCI-e add-in card can draw a maximum of 75 W through the standard connector, as specified in PCI Express CEM 1.1. GPU manufacturers provide extra power to the GPU by now using 8-pin (6-pin) wire-to-board connector that provide up to 150 W (75 W) additional power. This leads up to 300 W for one GPU card.

2.1 Description of Nvidia GPUs

In this paper we consider GPUs that are compatible with the DirectX 10 standard which provides a unified architecture where vertex and pixel shaders share the same instruction set and processing units. In order to efficiently exploit data parallelism, GPUs include a large number of replicated copies of these units operating in a single-instruction multiple-data (SIMD) fashion. GPUs handle high-latency instructions such as memory accesses by overlapping them using thousands of threads.

The unified architecture proposed by the DirectX 10 standard has been implemented in hardware since the release of the NVIDIA GeForce 8. An example of this architecture is depicted in Figure 1. The GPU has its own memory which bandwidth is usually an order of magnitude higher than system memory bandwidth. The graphics processor is seen as a set of *multiprocessors*. Each multiprocessor consists in numerous processing elements (PEs) operating in SIMD. At each



(a) Memory diagram. (b) Memory hierarchy.

Fig. 1. Diagram of G92 architecture

clock cycle, all the PEs in a multiprocessor execute the same instruction sequence, but operate on different data. These multiprocessors incorporate various types of memory such as a register file, a scratchpad memory shared by all the PEs in a SIMD block, and read-only constant cache and texture cache. In addition, PEs can also read or write global memory available on the graphic card.

The GeForce 8 has a shared cached constant memory accessible in broadcast mode only, a global memory accessible with coalescing (with additional alignment constraints), and a shared memory that allows coalesced, broadcast, and other patterns. Memory accesses that do not match these patterns are replaced by as many serial accesses as necessary, resulting in decreased performance. For example, the Tesla C870 embeds 1.5 GB of global memory with a peak bandwidth of 77 GB/s, which is more than 10 times the bandwidth available between a CPU and system memory. Multiprocessors integrate various computational units in order to implement the functionalities offered by the shaders: general computational units with embedded multiply-accumulators, texturing and filtering units, a dedicated unit to evaluate mathematical functions (e.g., sine, cosine, reciprocal, and reciprocal square root), and attribute interpolation units. Computational units can handle integer and floating-point arithmetic and there is no overhead associated with mixing both operations. Each multiprocessor of the GeForce 8 is able to execute a warp of 32 floating-point additions, multiplications, multiply-and-adds or integer additions, bitwise operations, comparisons, minimum or maximum of 2 numbers in 4 clock cycles. As there is no 32-bit integer multiplication in hardware, evaluating such an operation requires 16 clock cycles for a warp. For instance, one GPU of the GeForce 9800GX2, depicted in Figure 1, has 16 multiprocessors and each SIMD block is composed of 8 PEs and 2 functional evaluation units.

3 Measurement and Tests Description

We measure the power consumption of several Nvidia graphics card supporting the CUDA API described in Table 1. This includes a Tesla C870 card used for

Table 1. Main characteristics of the tested graphic cards. Frequencies of the GPU, computational units and memory are given, as well as the manufacturing process, the number of transistor and the temperature provided by the integrated sensor.

GPU	Commercial name	Core freq. (MHz)	Computing freq. (MHz)	Memory freq. (MHz)	Fab. Process (nm)	# of trans. $\times 10^9$	Temp. (°C)
G80	Tesla C870	575	1350	800	90	0.681	58
G92	GeForce 9800 GX2	600	1512	1000	65	2×0.754	61
GT200	T10P Prototype	900	1080	900	65	1.4	39

scientific computing, a dual GPU high-end graphics card 9800 GX2 and a T10P compute processor engineering sample.

The considered GPUs are tested in a desktop environment with a MSI X48 Platinum motherboard, an Intel Core 2 Duo E8400 processor with 6 MB L2 cache running at 3 GHz, four 1GB Corsair TWIN3X2048-1333C9DHX DDR3-1333 9-9-9-24 memory modules, and a 750 W Corsair HX750W power supply. This configuration is described in Figure 2. Our motherboard based on an X48 Intel chipset offers two PCI-e x16 2.0 and two PCI-e x4 1.0 slots through a bridge. The tested GPUs are included in a separated Tesla D870 box connected to the system through a dedicated bridge (NForce 100). Our software configuration uses Linux Ubuntu 8.04, with CUDA 2.0 and proprietary graphic drivers Nvidia 177.13.

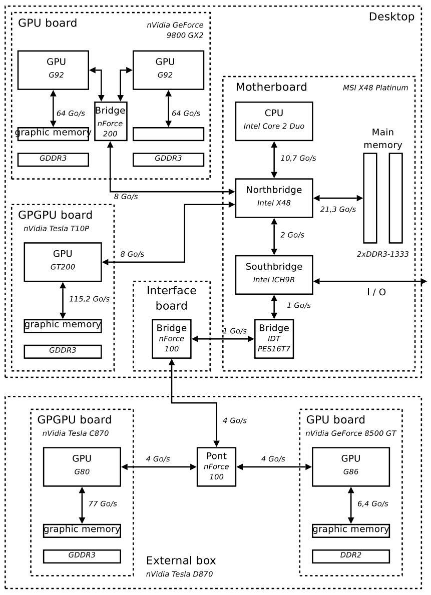


Fig. 2. Overview of our test configuration

To measure the power consumption we use a Tektronix TDS 3032 oscilloscope with the built-in analogue 20 MHz low-pass filter. The first input measures the current with a clamp sensor CA60, while the second measures voltage. The product of both measurements gives us the power consumption of the GPU through the external power. We took into consideration the power provided to the PCIe bus as well as the external power of the GPU. We noticed during our tests that the main source of power is provided by the external link, as during computation only 10 to 15 extra W are coming from the PCI bus.

We are aware that the measurements collected using this methodology are subject to caution. First, we not only measure the power consumption of the GPU but we also measure the power consumption of the whole board that includes the GPU, the graphic memory, DC/DC voltage converters, and others ICs (bridge, video chipset, . . .) However we assume that we will notice a difference in the power consumption only if these parts are stressed. Secondly, we measure the consumption ahead of the board voltage regulators which include small capacitors (decoupling). This means that the power consumption may be missestimated. We design short tests such that some variations in ambient temperature or the cooling system will not affect the results. Sampling rate is 50 kHz which allows us to measure the power consumption at task level but not at instruction level.

4 Results

4.1 Global Power Consumption

We measure the computation time and the average power required by common GPGPU algorithms (matrix transposition, matrix multiplication and cuBLAS) on 1024×1024 random matrices. The implementations of GPGPU algorithms used for these tests are the ones included in the Nvidia CUDA SDK 1.2. Results are reported in Table 2.

We observed during our tests that before the Linux driver was loaded, the GT200 was using around 20 W, which corresponds to the power saving mode, as officially claimed by Nvidia. Once the driver is loaded, the power consumption of the GT 200 rises to 51 W while idle. Surprisingly, we noticed that even though the

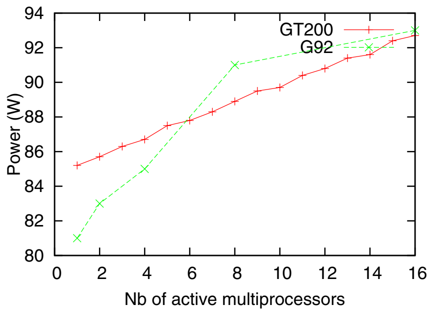
Table 2. Average power consumption P_{Avg}, computation time T and corresponding energy E measured for the execution of common GPGPU algorithms (Naive and optimized matrix transposition, matrix multiplication, cuBLAS)

GPU	Idle	Trans. naive			Trans. optimized			MatMul			cuBLAS sgemm		
		P_{Avg}	T	E	P_{Avg}	T	E	P_{Avg}	T	E	P_{Avg}	T	E
	(W)	(W)	(ms)	(J)	(W)	(ms)	(J)	(W)	(ms)	(J)	(W)	(ms)	(J)
G80	68	103	2.4	0.247	127	0.30	0.038	132	25	3.3	135	11.8	1.59
G92	71	94	3.66	0.344	105	0.45	0.047	117	23.3	2.73	122	11.4	1.39
GT200	51	73	4.11	0.300	83	0.50	0.042	114	13	1.48	113	7.44	0.84

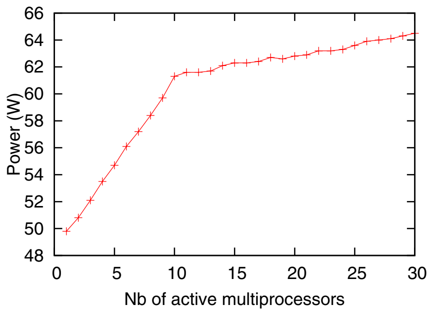
GT200 includes low-power GDDR3 compared to the Tesla C870 and a smaller fabrication process, the GT200 requires more energy than the Tesla C870 to perform a naive or an optimized matrix transposition. As the energy depends on memory access patterns on data which are located in memory bank modulo the bus width, the example choosen (matrix of size 1024) is in favor of the C870 with a bus width of 384 bits compared to the 512 bits of the GT200 that raise bank conflicts when accessing data. Nevertheless, the GT200 requires twice less energy than the G80 on the matrix multiplication or cuBLAS example. This suggests that memory access patterns play a major role from performance as well as power consumption perspectives.

4.2 Multiprocessor

Considered GPUs integrate several multiprocessors with their own front-end: instruction fetch, decode, issue logic and execution units, etc. We run tests where the number of active multiprocessors varies from 1 to the maximum (16 for G80 and G92 and 30 for GT200). Results are reported in Figure 3. We observe that power consumption rise linearly with the number of multiprocessors up to respectively the maximum, one half and one third of the available multiprocessor for the G80, G92, GT200. This implies that the scheduling strategy for blocks to execute is different between the G80 and the G92 / GT200. We can deduce that the scheduling strategy for G92 and GT200 for blocks execution is to execute one block per group of texture processors which includes 2 multiprocessors for the G92 and 3 multiprocessors for the GT200. This means that the scheduling strategy of the G92 and GT200 is optimized for bandwidth usage, whereas the one used in the G80 is optimized for energy saving when the executed kernel is computationally bounded.



(a) G80 - G92



(b) GT200

Fig. 3. Power variations depending on the number of active multiprocessors

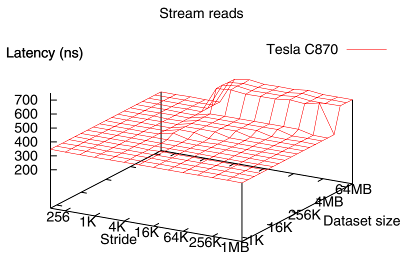
4.3 Execution Units

We measure the power consumption of various units within the GPU. We compare the power consumption of instructions predicated to true (that are really executed) and instructions that are predicated to false and we notice a significant variation. This means that instructions predicated to false are fetched, decoded but are not executed. This is unlike scalar architectures such as ARM or IA-64 behave, were predication is used to avoid pipeline stalls. On GPUs, predication is only used for SIMD control flow, while pipeline stalls are avoided using multi-threading. We measure the power used with various combination of units (MAD units, MAD and MUL units, MAD and complex function evaluation units and double-precision unit). Results are reported in Table 4.

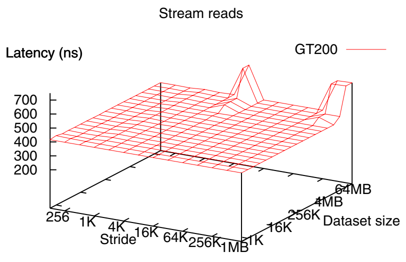
4.4 Memory Hierarchy

We measure how the memory hierarchy impacts the power consumption of GPUs. To measure the consumption, we developed a pointer chasing benchmark to access each level of the memory hierarchy. This benchmark includes a loop that traverses an integer array with various strides. 3 consecutive reads of one block of 128 bytes that will exploit coalescing are measured. Time is measured using the internal cycle counter of one multiprocessor. Only one warp on one multiprocessor is used. The size of the array and the stride vary. In order to minimize measurement noise, the minimum values among 10 executions are reported. Results for memory latency are given in figure 4.

The observed latency for each GPU varies between 300 and 800 ns. In Figure 4.a (C870), we observe 3 steps at respectively 345 ns, 372 ns and 530 ns depending on the locality of the access. G92 behaves the same way with respective latencies of 325, 350 and 500 ns. We believe that these variations come from TLB hits and misses. Therefore, these results suggest a size of 4 KB per page with a 32-entry fully associative TLB. Results suggest that the second TLB level is accessed in 16 cycles on the G80 and G92 and 18 cycles on the GT200. However, our tests do not allow us to determine if it consists in an actual TLB or of a page table entry (PTE) cache. It is possible that it is shared with the second level of

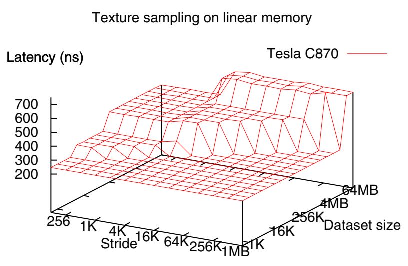


(a) Tesla C870

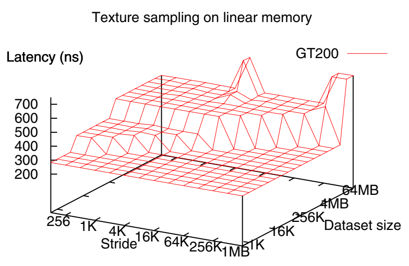


(b) GT200

Fig. 4. Latency for one read in global memory



(a) Tesla C870



(b) GT200

Fig. 5. Latency for one read in texture memory

Table 3. Energy per memory request depending on where data is located. Energy results (nJ) per memory request of 128 bytes correspond to the measured power consumption divided by the measured bandwidth.

Memory access	G80	G92	GT200
Coalesced stream read	124.4	103.4	80.6
L1 texture	60.7	28.3	25.6
L2 texture	62.3	48.0	66.7
Texture miss	102.2	110.6	83.1

constant cache included in each texture processor. This cache may be used for instructions, constants, and PTEs.

DRAM is divided into pages that need to be precharged before each access and subsequently unloaded. According to our tests, DRAM page misses do not impact measured latency and power consumption. This may come from the fact that the memory controller optimizes page activation and unloading according to the address and type of memory access waiting in the buffer and that the access latency is made constant based on the worst case to simplify the design of the memory controller.

The latencies of texture memories are given in Figure 5. We observe the same steps as in Figure 4 plus the variation due to the two texture caches. By comparing the latency of a cache miss in the texture cache with a memory read in the same conditions, we estimate at 30 to 50 cycles the overhead due to texture filtering and the extra cache access depending on the GPU generation. We also measured that the latency to access the L2 cache and to go through the memory crossbar should be between 35 and 50 cycles.

We measured the energy required for one memory request of 128 bytes executed by a warp on the G80, G92 and GT200 architectures. Results are reported in Table 3. The G80, G92 and GT200 are respectively made of 16, 16 and 30 multiprocessors. Kernels used for the tests are launched with 512 threads/block with

Table 4. Number of instructions, measured power P, number of cycles per instruction and the energy required per warp for ALU operation on G80, G92 and GT200

		G80			G92			GT200		
Operation	# inst.	P (W)	CPI	E (nJ/warp)	P (W)	CPI	E (nJ/warp)	P (W)	CPI	E (nJ/warp)
MAD	32	107	4.75	8.57	100	4.29	5.06	91	4.3	5.31
Pred	32	90	2.38	2.43	93	2.39	2.14	75	2.36	1.75
MAD+MUL	64	117	3.19	7.24	111	2.83	4.61	102	2.82	4.44
MAD+RCP	40	115	3.96	8.63	110	3.55	5.63	98	3.54	5.14
RCP	8	98	15.89	22.07	96	16	16.28	81	15.99	14.81
MOV	32	118	2.31	5.34	113	2.46	4.21	101	2.46	3.79

as many blocks as multiprocessors. The instruction count reflects the number of operations that can be executed in a single warp. The measured power corresponds to the average power measured during the whole computation. CPI gives the number of clock cycles per instruction which corresponds to the number of clock cycles measured with the internal cycle counter divided by the number of threads per block. The energy per warp corresponds to extra power required to perform a specific operation for a warp multiplied by the computational time. We observe that the L1 texture cache of the G80 requires the same amount of energy per byte as the L2 texture cache. This is certainly due to the small bandwidth performance of the L1 texture cache (76 GB/s for L1 and L2 texture cache). However we cannot explain the small performance of the L2 texture cache of the GT200. The comparison of Table 3 and 4 leads to the conclusion that executing MAD is 7 to 15 times more energy efficient than accessing memory on a G80.

5 Conclusion

This article presents some measurements and an analysis on how computations and memory accesses impact the power consumption of some Nvidia GPUs (G80, G92, GT200) in the CUDA framework. Memory accesses can significantly degrade the performance and power consumption such that the G80 may be a better alternative than the latest GT200 in some specific cases. Our tests show how blocks of threads are dispatched among the multiprocessors when the number of multiprocessors is larger than the number of blocks and how it could negatively impact the power consumption for computationally bounded kernels. We also compare the energy required for various operations for a given warp with the energy required for a memory access and show that computations are up to 7 to 15 times more power efficient than memory accesses.

Our tests show that environmental conditions such as temperature and measurement equipment can impact the results by 10 to 15%. Therefore we are currently working on minimizing the impact of the environment by executing the same tests with others tools in a different environment. We also plan to measure the clock gating impact and perform comparisons between Nvidia and ATI GPUs for a given workload.

References

1. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. SIGARCH Comput. Archit. News 28(2), 83–94 (2000)
2. Piguet, C. (ed.): Low-Power Electronics Design. Computer Engineering, vol. 1941. CRC Press, Boca Raton (2004)
3. Sasanka, R., Adve, S.V., Chen, Y.-K., Debes, E.: The energy efficiency of cmp vs. smt for multimedia workloads. In: ICS 2004: Proceedings of the 18th annual international conference on Supercomputing, pp. 196–206. ACM, New York (2004)
4. Senn, E., Laurent, J., Julien, N., Martin, E.: SoftExplorer: Estimating and optimizing the power and energy consumption of a C program for DSP applications. EURASIP Journal on Applied Signal Processing, 2641–2654 (January 2005)
5. Weste, N.H.E., Harris, D.: CMOS VLSI Design: A Circuits and Systems Perspective, 3rd edn. Addison Wesley, Reading (2005)
6. Ye, W., Vijaykrishnan, N., Kandemir, M., Irwin, M.J.: The design and use of simplepower: a cycle-accurate energy estimation tool. In: DAC 2000: Proceedings of the 37th conference on Design automation, pp. 340–345. ACM, New York (2000)

Experiences with Mapping Non-linear Memory Access Patterns into GPUs

Eladio Gutierrez, Sergio Romero, Maria A. Trenas, and Oscar Plata

Department of Computer Architecture
University of Malaga, Spain
{eladio,sromero,maria,oplata}@uma.es

Abstract. Modern Graphics Processing Units (GPU) are very powerful computational systems on a chip. For this reason there is a growing interest in using these units as general purpose hardware accelerators (GPGPU). To facilitate the programming of general purpose applications, NVIDIA introduced the CUDA programming environment. CUDA provides a simplified abstraction of the underlying complex GPU architecture, so as a number of critical optimizations must be applied to the code in order to get maximum performance. In this paper we discuss our experience in porting an application kernel to the GPU, and all classes of design decisions we adopted in order to obtain maximum performance.

1 Introduction

Driven by the huge computing demand of the graphics applications, Graphics Processing Units (GPU) have become highly parallel, multithreaded and many-core processors. Modern GPUs deliver a very large amount of raw performance that have drawn attention to the scientific community, with a growing interest in using these units to boost the performance of their compute-intensive applications. That is, to use the GPUs as general-purpose hardware accelerators (General-Purpose Computation on GPUs, or GPGPU [2]).

Developing GPGPU codes using the conventional graphics programming APIs is a very hard task and with many limitations. This situation motivated the development of general parallel programming environments for GPUs [11,12]. NVIDIA CUDA (Compute Unified Device Architecture) [11], one of the most widespread models, is built around a massively parallel SIMT (Single-Instruction, Multiple-Thread) execution model, supported by the NVIDIA GPU architecture [7], and provides a shared-memory, multi-threaded architectural model for general-purpose GPU programming [10].

CUDA provides a convenient and successful model at programming scalable multi-threaded many-core GPUs, across various problem domains [5]. However, the simplified abstraction that CUDA model provides does not permit to extract maximum performance from the underlying GPU physical architecture without applying a set of optimizations to the parallel code [8,13]. We can distinguish two classes of optimizations. The first class corresponds to techniques that fall within the programming model, that is, those that improve the use of the architectural resources defined at CUDA level. The second class includes those optimizations that fall outside the programming model. We consider in this class techniques at a level lower than CUDA, that is, that must be included in the parallel execution implementation of the programming model.

This paper discusses our experience in porting application kernels to a GPU accelerator, with the aim of obtaining maximum performance. In order to have enough room for optimization, we have selected as a working example a kernel showing non-linear access patterns to memory, the fast Fourier transform (FFT). We will show that if the optimization efforts are only within the CUDA model, the obtained performance is much lower than the expected peak one. We have to resort to additional low-level techniques in order to improve significantly the resulting performance. An important issue is that these techniques are hard to apply and very dependent on the kernel computational structure. A final issue we also analyzed refers to the algorithm chosen to implement the kernel.

Due to its interest, several contributions can be found in the literature focused on porting FFT algorithms to graphics processing units [1,3,6,9,14]. More recently works about CUDA implementations report higher performance [4,15]. This is accomplished by a much more efficient use of GPU resources, through the application of many optimization techniques (CUDA level and low level). The implementation described in [4] behaves specially well. They use a different algorithm for FFT, a hierarchical Stockham.

2 CUDA Programming Model

NVIDIA CUDA is both a hardware and software architecture for issuing and managing computations on the GPU, making it to operate as a truly generic data-parallel computing device. An extension to the C programming language is provided in order to develop source codes.

From the hardware viewpoint, the GPU device consists of a set of SIMT multiprocessors each one containing several processing elements. Different memory spaces are available. The global device memory is a unique space accessible by all multiprocessors, acting as the main device memory with a large capacity. Besides, each multiprocessor owns a private on-chip memory, called shared memory or parallel data cache, of a smaller size and lower access latency than the global memory. A shared memory can be only accessed by the multiprocessor that owns it. In addition, there are other addressing spaces for specific purposes.

CUDA execution model is based on a hierarchy of abstraction layers: grids, blocks, warps and threads. The thread is the basic execution unit that is actually mapped onto one processor. A block is a batch of threads cooperating together in one multiprocessor and hence all threads in a block share the shared memory. A grid is composed by several blocks, and because there can be more blocks than multiprocessors, different blocks of a grid are scheduled among the multiprocessors. In turn, a warp is a group of threads executing in an SIMT way, so threads of a same block are scheduled in a given multiprocessor warp by warp.

Two kinds of codes are considered in the CUDA model: those executed by the CPU (host side) and those executed by the GPU, called kernel codes. The CPU is responsible of transferring data between host and device memories as well as invoking the kernel code, setting the grid and block dimensions. Memory accesses and synchronization schemes are the most important aspects to take into account. Warp addresses issued by SIMT memory access instructions may be grouped thus obtaining a high memory bandwidth. This is known as coalescing condition. Otherwise, access will be serialized and the resulting latency will be difficult to hide with the execution of other warps of the same block.

3 Experiences in Optimizing the FFT in CUDA

We have selected as a benchmark a kernel code showing non-linear access patterns to memory: the Fast Fourier Transform (FFT). Basically, the FFT follows a *divide and conquer* strategy in order to reduce the computational complexity of the discrete Fourier transform (DFT), which provides a discrete frequency-domain representation $X[k]$ from a discrete time-domain signal $x[n]$. For a 1-dimensional signal of N samples, DFT is defined by the following pair of transformations (forward and inverse): $X = DFT(x)\colon\ X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk},\quad 0 \leq k < N$, and $x = IDFT(X):\ x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn}, 0 \leq n < N$, where the powers of $W_N = e^{-j\frac{2\pi}{N}}$ are the so-called twiddle factors.

The design decisions to develop an efficient GPU implementation of a kernel code like FFT may be classified into three levels:

– *Algorithm level*: It refers to the algorithm chosen to implement the kernel. The basic strategy is the well-know Cooley-Tukey decomposition, but some other strategies, like the Stockham approach, have been shown to behave better in SIMD architectures. In addition, the selection of the radix parameter has strong influence in the performance.
– *CUDA level*: Once the algorithm has been configured, it must be mapped into the CUDA architecture. The resulting performance depends strongly on two main issues: parallelism extracted and memory hierarchy exploitation. Both issues are closely related to platform features, and are clearly influenced by the problem memory access patterns.
– *Code level*: The CUDA architecture hides many low-level details of the underlying hardware platform. This way, an important fraction of the overall performance may depend on a series of low level tricks that would help the compiler to generate an efficient object code.

Algorithm level. In this paper we have considered a radix-2, DIT (Decimation In Time), Cooley-Tukey implementation. Although this algorithm requires an initial bit-reversal stage, however it could be used in signal transformations where such bit-reversal operation is not required (e.g., Walsh). This is not the case of the Stockham method, which is auto-sorted.

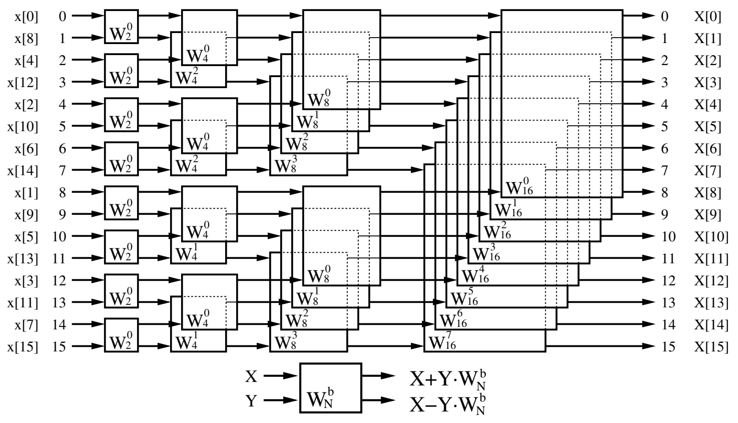


Fig. 1. Radix-2 decimation-in time FFT in terms of butterfly operators

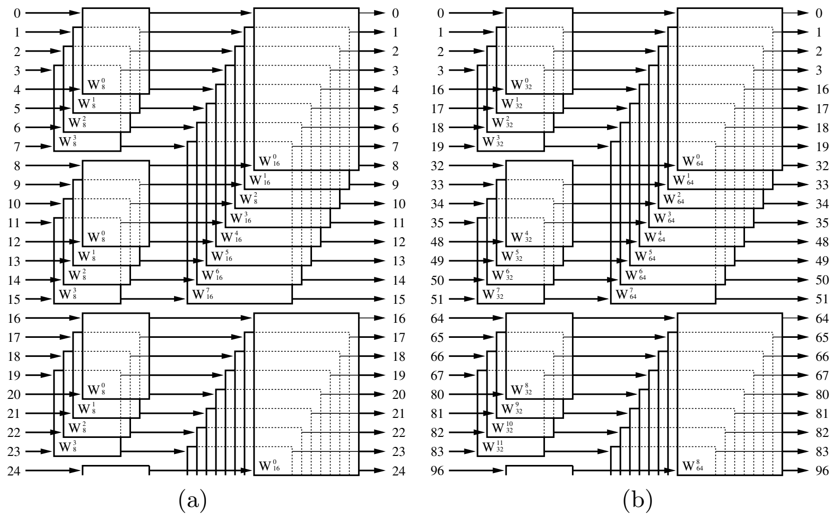


Fig. 2. (a) Computing 3rd and 4th stages of the FFT; (b) computing 5th and 6th stages of the FFT using the pattern of 3rd and 4th stages over a properly permuted input

The radix-2, DIT, Cooley-Tukey FFT organizes the DFT computations, as shown in Fig. 1, in terms of basic blocks, known as butterflies. The computation is carried out along $\log_2 N$ stages being computed N coefficients per stage. Before the first stage, input coefficients must be bit reversed (omitted in the figure).

Focusing on memory locality, we observe that if the input coefficients are located into consecutive memory positions, the reference patterns of higher stages will exhibit poorer locality features than the lower ones. In addition, if the input coefficients are permuted properly, it is possible to carry out one of the stages using the access pattern of another, simply by using the corresponding twiddle factors. Such an equivalence is depicted in Fig. 2 showing how 5th and 6th

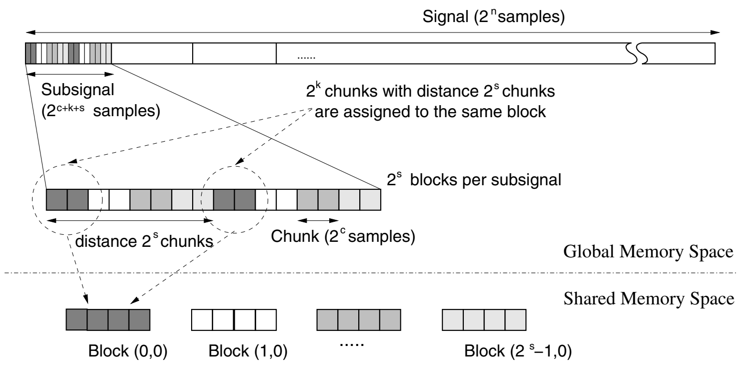


Fig. 3. Mapping of the input signal from global to shared memory spaces

stages can be performed with the access pattern of the 3^{rd} and 4^{th} ones, after permuting the coefficients. This mapping is denoted as: $[5 : 6] \rightarrow [3 : 4]$.

CUDA level. The goal in the CUDA version is to obtain a high degree of parallelism taking into account system constrains, specifically those related to the memory hierarchy. The basic idea consists of mapping input signal samples placed in global (device) memory into the data parallel cache (shared memory), performing all possible computations with these local data and then copying the updated samples back to the global memory. This process may be repeated with different mapping functions until all stages are done.

Fig. 3 depicts how the input signal is repeatedly mapped from global memory into shared memory spaces. This mapping try to maximize coalesced accesses to global memory. The figure also shows a number of parameters defined to describe our CUDA version, named as ctFFT (from Cooley-Tukey FFT) from now on. First, we consider that the input signal size is a power of two (2^n samples). This signal is subdivided into equal-sized subsignals, which are further subdivided into fixed-size chunks of 2^c samples. A set of 2^k chunks, separated among themselves a distance of 2^s chunks, are grouped and assigned to the same CUDA block. So, each subsignal contains a total of 2^s chunk blocks, or a total of $2^{(s+k+c)}$ samples. Hence, the size of each CUDA block is of $2^{(c+k)}$ samples, and the complete input signal contains a total of $2^{n-(c+k)}$ such blocks. There is a size restriction for CUDA blocks, as each one will be processed in parallel in a single GPU multiprocessor, so it must be fitted completely in the shared memory.

With the above data mapping strategy, ctFTT proceeds as a series of synchronized phases, as follows:

– **Initial Phase**: Processing of CUDA blocks composed of consecutive chunks ($s = 0$). The first $k + c$ FFT stages can be accomplished with these data blocks ($[0 : (k + c - 1)] \rightarrow [0 : (k + c - 1)]$).
– **Intermediate Phase 1**: After finishing these first $k + c$ stages, we can continue with the remaining FFT stages. These stages should not overlap

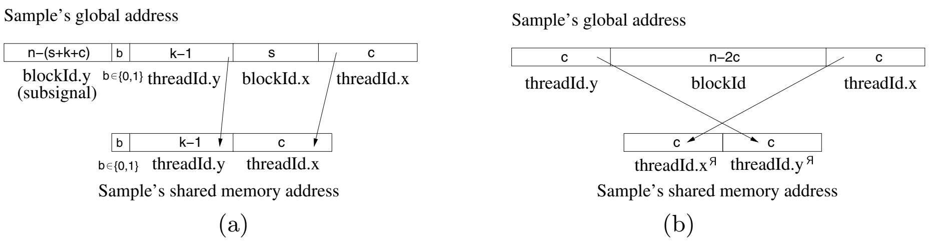


Fig. 4. (a) Bit-block addressing defining the mapping of the input signal from global to shared spaces, and (b) addressing for the bit reversal operation

the previous ones already processed, so we must select the suitable CUDA blocks to be transferred to shared memory. Stage overlapping is avoided if we select blocks corresponding to values of s that are integer multiple of k. So, in the first intermediate phase, we process CUDA blocks composed of chunks separated k chunks, that is, $s = k$. These data allow to compute the following k FFT stages: $[s + c : s + c + k - 1] \rightarrow [c : c + k - 1] = [k + c : 2k + c - 1] \rightarrow [c : c + k - 1]$.

– **Intermediate Phase i**: In general, the above procedure is repeated. Now, we take CUDA blocks corresponding to $s = ik$, that allow to compute a bunch of k FFT stages: $[s + c : s + c + k - 1] \rightarrow [c : c + k - 1] = [ik + c : (i + 1)k + c - 1] \rightarrow [c : c + k - 1]$.
– **Final Phase**: The last FFT stages to be computed could be less than k. If the total number of intermediate phases is P, in this final phase the next FFT stages are computed: $[s + c : n] \rightarrow [c : n - s] = [(P + 1)k + c : n] \rightarrow [c : n - (P + 1)k]$. The number of intermediate phases can be calculated as $P = \lfloor (n - (k + c))/k \rfloor$, if this number is positive.

ctFFT organizes parallel execution by assigning to each thread two main tasks: (i) copying of two signal samples from global to shared spaces, and (ii) processing of a single FFT butterfly, accessing two signal samples stored in shared memory. So, the total number of threads is $2^n/2 = 2^{n-1}$ (radix-2). These threads are organized in a grid of $(nBlock.x, nBlock.y)$ thread blocks, and each of these thread blocks groups $(nThreads.x, nThreads.y)$ threads. With this grouping of threads, the data mapping shown if Fig. 3 can be defined by mapping bit-blocks of the memory addresses, as depicted in Fig. 4 (a). The b bit in both addresses allows to distinguish between the two signal samples assigned to the same thread, and it is in use during the copy-in and copy-out operations (global to shared and shared to global). During the computation of a butterfly in the i FFT stage, the thread accesses the corresponding signal samples in shared memory by inserting a bit 0 or 1 in the i bit of the shared memory address composed of the bit-blocks $threadIdx.y|threadIdx.x$. Note that if c is larger than the number of threads in a single warp, then coalescing condition is completely fulfilled.

As a DIT implementation is considered here, the initial bit reversal operation applied to the input signal is required. Fig. 4 (b) shows how this operation is implemented in ctFFT. Basically, it consists in a coalesced copy of the input signal samples to the shared memory, storing them in positions given by the bit reversal of the thread bit-block identifiers. Afterwards, these samples are copied back to the corresponding positions in global memory (not in-place).

Code level. Among the low-level optimization techniques, we can highlight four with high impact on the code performance: loop unrolling, padding, constant propagation and thread synchronization. Padding is used to reduce shared memory bank conflicts. Constant propagation avoids unnecessary arithmetic instructions, specially when computing padding functions. Additional non-mandatory thread barrier synchronizations are beneficial. For example, after completing global memory accesses. We consider that most of these optimizations should be implemented in the CUDA compiler, but our experience shows that without the help of the programmer, the compiler fail to apply many of these techniques.

4 Experimental Evaluation

In this section we experimentally evaluate ctFFT. All experiments were conducted on a NVIDIA GeForce 280 GTX GPU, which includes 30 multiprocessors of eight processors each (240 cores in total), working at 1.3GHz with a device (global) memory of 1 GB. Each multiprocessor has a 16KB parallel data cache (shared memory). Codes were written in C using the version 1.0 of NVIDIA CUDA [11]. NVIDIA provides its own FFT library (CUFFT), that we take as a reference in order to assess the quality of our optimized CUDA version (ctFFT).

Fig. 5 (top) shows the performance of ctFFT compared to CUFFT. These plots correspond to the CUDA version discussed in the previous section, considering only CUDA level optimizations, specially, coalescing (no low-level). According to the CUFFT interface, two dimensionality parameters are taken into consideration: the signal size and the number of signals of the given size to be processed (known as a batch of signals). The number of FLOPS is calculated using the equation $5bN \log_2 N$, for a batch of b signals of N samples per signal.

From the plots it can be seen that in cases of batches of large signals, our ctFFT outperforms CUFFT. However, CUFFT is better for a large number of batches of small signals. In addition, ctFFT allows to process larger signals than CUFFT. The CUFFT library is unable to perform the transform beyond 2^{23} samples [11] whereas our implementation can manage up to 2^{26} samples, making a better exploitation of the available device memory. Fig. 5 bottom summarizes all these results (GFLOPS in the plot at the left, relative GFLOPS in the plot at the right). The bottom-right plot allows to determine for which signal configurations ctFFT outperforms CUFFT.

The above results show that the best performance attained is almost 40 GFLOPS, which is much lower than the peak performance of the GPU platform. And there are no other relevant performance strategies at CUDA level that we can use to further improve the parallel code. So, only two alternatives remain, either change the original FFT algorithm, that maps better into the

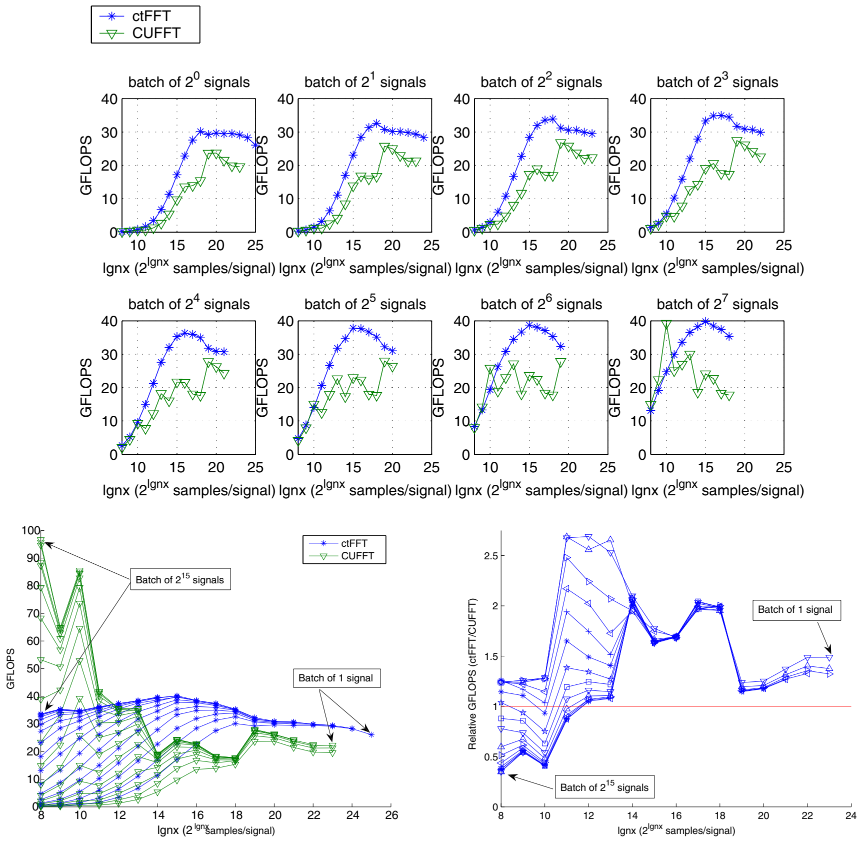


Fig. 5. Performance in GFLOPS of ctFFT compared to CUFFT

GPU architecture, or apply low-level optimization techniques. These optimizations fall outside the CUDA programming model, are dependent on the specific CUDA code at hand, and represent a hard effort to apply.

Table 1 (a) illustrates the change in performance that ctFFT underwent when an incremental series of low-level optimization techniques were applied. These figures were obtained for a batch of 2^{15} signals of 2^9 samples per signal, in such a way that each signal fits completely in the shared memory of a CUDA block. This table shows a broad range of achieved performance results, from 17.5 to 229 GFLOPS, depending on different optimizations and algorithms used. At present, the best known FFT implementation on CUDA [4] performs a peak of 300 GFLOPS (based on considerable hand-coded low-level optimizations). The first column in the table corresponds to a ctFFT version where all computations are carried out over the global memory, using coalesced accesses (shared memory is not used). The second column is the CUDA version analyzed in the previous section. This version uses the shared memory as a cache of the global one, resulting in about 2× performance improvement. The next two columns

Table 1. Incremental performance improvements: (a) applying low-level optimizations, (b) for different memory access patterns (batch of 2^{15} signals of 2^9 samples)

(a)

	ctFFT (global)	ctFFT	+ unrolling	+ padding	+ radix-4	+ radix-8	+ constant propagation
GFLOPS	17.5	35.1	59.5	63.5	106	116	130
Incoherent ld/st	0	0	0	0	0	0	0
Warp serialize	0	$806 \cdot 10^3$	$668 \cdot 10^3$	$649 \cdot 10^3$	$428 \cdot 10^3$	$255 \cdot 10^3$	$221 \cdot 10^3$

(b)

	DIT, no bit reversal	DIF, wo/ synch	DIF, w/ synch	Stockham
GFLOPS	142	167	199	229
Incoherent ld/st	0	0	0	0
Warp serialize	$189 \cdot 10^3$	$211 \cdot 10^3$	$170 \cdot 10^3$	$95 \cdot 10^3$

add loop unrolling and padding low-level optimizations to ctFFT. The performance improvement due to padding is lower than expected because the padding function must be simple in order to not introducing too much overhead. The use of a higher radix (than 2), as shown in the next two columns, allows to increase the performance even more. The reason for this behaviour is a more intensive (re)use of the processor registers, that represent the fastest level of the memory hierarchy. For radices beyond 8, the performance degrades due to two effects. First, the fixed number of registers limits the number of active threads per multiprocessor (this is called occupancy). Second, for a higher radix, the number of threads per block decreases, reducing the opportunities of hiding memory latency via warp scheduling. Constant values resulting from padding functions can be precomputed and propagated directly in the code. This optimization has an important impact in the performance, as shown in the corresponding column.

Table 1 (b) corresponds to other four implementations we have developed for the same signal configuration, and also including all the previously discussed low-level optimizations. This table show the impact in performance of various memory access patterns. The first one is ctFFT but with the bit reversal stage omitted. The second and third columns correspond to the radix-8, DIF (Decimation In Frequency) version of the Cooley-Tukey algorithm. The difference between them is the inclusion or not of additional non-mandatory thread barrier synchronizations. Finally, the fourth column corresponds to the Stockham algorithm, developed using similar strategies than discussed for ctFFT.

Basically, the performance of ctFFT is modest due to two main reasons: the cost of the bit reversal operation, and the loss of parallelism due to conflicts in shared memory banks. Other interesting measurements included in tables are the number of incoherent loads/stores and the warp serialize parameter (obtained by activating `CUDA_PROFILE`). The first one shows the number of non-coalesced accesses to global memory. In all evaluated cases, coalescing was perfect. Warp serialize represents the number of threads serialized due to shared memory bank conflicts. Note that the low-level optimizations reduces significantly this number, with a clear effect in performance. But also the memory access pattern associated to the algorithm (Cooley-Tukey/Stockham, DIT/DIF, radix-2/-4/-8) has an important impact. The lowest number of warp conflicts correspond to Stockham version (radix-8, DIF).

5 Conclusions

This paper discusses our experience in porting application kernels to GPU accelerators using CUDA. In particular, a FFT benchmark was chosen due to its non-linear memory access patterns that allow to play with many design issues when mapping it to the complex GPU architecture. According to our experience, developers should take into account three classes of design issues: at algorithm level, at CUDA level and at code level (low level). In any level we may observe strong impact in performance. We specially highlight low-level optimizations, that may increase the performance of the CUDA version by one order of magnitude. However, the bad part is that these techniques are very dependent on the specific CUDA code and represent a hard effort to apply. New programming environments should include technology to automatize, at least partially, these essential and high-impact low-level optimizations.

References

1. Fialka, O., Cadik, M.: FFT and Convolution Performance in Image Filtering on GPU. In: 10th Int'l. Conf. on Information Visualization (2006)
2. General-Purpose Computation Using Graphics Hardware, `http://www.gpgpu.org`
3. Govindaraju, N.K., Larsen, S., Gray, J., Manocha, D.: A Memory Model for Scientific Algorithms on Graphics Processors. In: ACM Int. Conf. Supercomputing (2006)
4. Govindaraju, N.K., Lloyd, B., Dotsenko, Y., Smith, B., Manferdelli, J.: High Performance Discrete Fourier Transforms on Graphics Processors. In: Int'l. Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2008) (2008)
5. Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., Volkov, V.: Parallel Computing Experiences with CUDA. IEEE Micro. 28(4), 13–27 (2008)
6. Jansen, T., von Rymon-Lipinski, B., Hanssen, N., Keeve, E.: Fourier volume rendering on the GPU using a split-stream FFT. In: Vision, Modeling, and Visualization Workshop (2004)
7. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro. 28(2), 39–55 (2008)
8. Manikandan, M., Bondhugula, U., Krishnamoorthy, S., Ramanujam, J., Rountev, A., Sadayappan, P.: A Compiler Framework for Optimization of Affine Loop Nests for GPGPUs. In: ACM Int'l. Conf. on Supercomputing (2008)
9. Moreland, K., Angel, E.: The FFT on a GPU. In: ACM Conf. Graph. Hardware (2003)
10. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable Parallel Programming with CUDA. ACM Queue 6(2), 40–53 (2008)
11. NVIDIA CUDA (2008), `http://developer.nvidia.com/object/cuda.html`
12. The OpenCL Specification, Ver. 1.0.29, Khronos OpenCL Working Group, `http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf`
13. Petit, E., Matz, S., Bodin, F.: Data Transfer Optimization in Scientific Applications for GPU based Acceleration. In: Workshop Compilers for Parallel Computers (2007)
14. Sumanaweera, T., Liu, D.: Medical Image Reconstruction with the FFT. GPU Gems 2, 765–784 (2005)
15. Volkov, V., Kazian, B.: Fitting FFT onto the G80 Architecture (2008), `http://www.cs.berkeley.edu/~kubitron/courses/cs258-S08/projects/reports/project6_report.pdf`

Accelerated Discovery of Discrete M-Clusters/Outliers on the Raster Plane Using Graphical Processing Units

Christian Trefftz[1], Joseph Szakas[2], Igor Majdandzic[1], and Gregory Wolffe[1]

[1] School of Computing, Grand Valley State University, Allendale, MI 49401
[2] Computer Information Systems, Univ. of Maine-Augusta, Augusta, ME 04330
{trefftzc,wolffe}@gvsu.edu, majdanig@student.gvsu.edu, szakas@maine.edu

Abstract. This paper presents two discrete computational geometry algorithms designed for execution on Graphics Processing Units (GPUs). The algorithms are parallelized versions of sequential algorithms intended for application in geographical data mining. The first algorithm finds clusters of *m* points, called m-clusters, in the rasterized plane. The second algorithm complements the first by identifying outliers, those points which are not members of any m-clusters. The use of a raster representation of coordinates provides an ideal data stream environment for efficient GPU utilization. The parallel algorithms have low memory demands, and require only a limited amount of inter-process communication. Initial performance analysis indicates the algorithms are scalable, both in problem size and in the number of seeds, and significantly outperform commercial implementations.

Keywords: GPU algorithms, Geographical data mining, CUDA programming.

1 Introduction

Traditional computational geometry algorithms are applied to subsets of the plane, using real values to designate point coordinates. Thus, there are an infinite number of points in the domain of the problem.

In Geographical Information Systems (GIS) the use of a raster representation defines a finite number of points (or pixels) in the problem domain. There are several advantages to using a raster of vector data for spatial processing: the raster data model is well-suited for GIS applications aimed at the analysis of the spatial variability of geographic phenomena, it is more effective in the representation of continuous surfaces, and it can be exploited for accelerated processing and display [1].

Recently, Gudmundsson, van Kreveld and Narasimhan [2] presented several computational geometry algorithms suitable for applications in geographical data mining. Their algorithms find clusters of points on the plane and identify outliers, assuming two types of land cover and defining a cluster via its bounding square. In this paper, we present parallelized algorithms to solve the same problems without these assumptions and restrictions. Furthermore, our algorithms are optimized to execute on Graphics Processing Units (GPUs). GPUs, in addition to being highly parallelized computational units, are particularly well-suited for spatial applications.

1.1 Graphics Processing Units

GPUs, ubiquitous on video cards and familiar to the gaming community, have been steadily increasing in speed, capacity and number of processing units. Given the insatiable demand of modern graphics applications, it seems reasonable to assume that they will continue to do so. Until recently, GPU usage was largely limited to the rendering expected of a video card. That has begun to change with the advent of General Purpose computation on Graphics Processing Units (GPGPU), and the new languages and environments developed in support of that goal. Languages used to program GPUs typically use the Single Instruction Multiple Data (SIMD) style of coding. A program specifies a sequence of operations that is to be applied to the vertices of a polygon or to the pixels comprising an image to be rendered. These operations execute in parallel on different processing units, each on their own unique vertex or pixel.

General purpose computation on GPUs extends this methodology to operate on non-rendering data. For example, one of the new GPU programming languages is called CUDA (Computer Unified Device Architecture) [3], designed for NVIDIA's GPUs. CUDA has a number of elements in common with other Data Parallel Languages, such as MPL [4], C* [5] and Parallaxis [6]. In CUDA, the programmer specifies a *kernel*, a piece of code that is executed in parallel on all of the processing elements in a GPU. During parallel execution, each processing element is responsible for one pixel (or data object), and the kernel contains precisely the set of operations that is to be applied to every element. Programmer-defined “blocksize” and “gridsize” attributes partition the data and direct its redistribution to processing elements.

CUDA's run time environment schedules the parallel execution of the code in the kernel on the multiple processing units inside the GPU. The local variables within a kernel belong to a particular instance of a kernel and each instance of the kernel has its own version (copy) of those variables. In some ways the kernel resembles a class; many instances of the kernel may be active at the same time, each with its own "instance" local variables.

Special operations allow the programmer to move data between main memory on the "host" computer and the device memory on the GPU card.

1.2 Computational Geometry / Geographical Data Mining

Designing and developing algorithms for GPU execution is an active research area. Some of these algorithms use graphic primitives to find approximate solutions to computational geometry problems. Given the nature of GPUs and the displays upon which solutions are ultimately rendered, it is natural to work in the domain of pixels, using integer coordinates. For example, Hoff et al. [7] describe an algorithm to calculate generalized Voronoi diagrams. Fan et al. [8] based their algorithm for natural neighbor interpolation on the same discretization of the plane. Denny [9,10] uses GPU primitives to implement algorithms that solve the problems of: determining the smallest homothetic scaling of a star-shaped polygon enclosing a set of points, finding the largest empty homothetic scaling of a polygon completely contained inside an arbitrary region, and adding a new point to an existing Voronoi diagram such that the resulting Voronoi region has maximal area.

The remainder of this paper is structured as follows: a parallelized algorithm for finding clusters of *m* points is described in section 2. Section 3 contains a modified algorithm for finding outliers. Initial performance results are presented in section 4. Finally, section 5 presents our conclusions.

2 Finding M-Clusters

Clusters can be defined in different ways [11]. One way is to think of a cluster as a "large enough" subset of points that are close to each other. In GIS, finding properties of point sets is known as point pattern analysis [12].

Based on this notion of a cluster, we present the following definitions: let *P* be a set of *n* points in the plane, representing the objects that are being analyzed for clustering. These *n* points are known as seeds. Let *r* denote a distance of interest. The set of points in a cluster should lie inside a circle of radius *r* or smaller. Let *m* denote the minimum number of points required to form a cluster. The problem then, is to identify subsets of *m* points (or more) that are inside a circle of radius *r* or smaller. We refer to this group of points as an m-cluster.

Gudmundsson's paper contains a detailed discussion of the issue of the precision of the results when one "snaps" the coordinates of the seeds to integer coordinates, and one of the algorithms presented in that paper uses integer coordinates for the seeds. The following example illustrates the integer approach.

Consider a raster surface of size 12 × 12 with 4 seeds located at the coordinates: (4,4), (5,10), (6,2) and (7,9). Assume that we have specified that we need $m = 3$ seeds per cluster and that the radius of every m-cluster should be $r = 4.2$ or smaller. For this particular set of input data, two points qualify as centers of 3-clusters: (6,5) and (6,6). In the raster surface of Figure 1, the '×' symbol denotes the locations of the seeds and '·' denotes the locations of the centers of m-clusters.

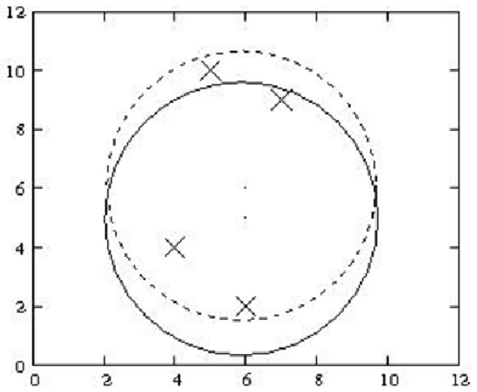


Fig. 1. An example showing two 3-clusters, one centered at (6,5) and another at (6,6)

Note that a seed may be a member of more than one m-cluster, and that an m-cluster may contain more than *m* seeds. Note also that more than one pixel may be a candidate center for the same m-cluster. The criterion for choosing a particular center for an m-cluster depends on the specific application. For example, it may be important to choose the candidate center pixel with the smallest resultant m-cluster radius.

2.1 Sequential M-Clustering Algorithm

We next describe a sequential algorithm that calculates the set of m-clusters for a set *P* of points. The area is represented by a raster surface of size xSize × ySize, which is the sub-plane of interest. The seeds are assumed to be inside the area covered by the raster surface. The algorithm looks for points on the raster surface that are the (approximate) centers of circles of radius *r* or smaller that contain at least *m* points, out of the original *n* points. The pseudo-code for this algorithm is given in Figure 2.

```
For every pixel x in the raster surface
   Initialize counter to 0
   For every seed s in the set of n seeds
      Find the distance d from pixel x to seed s
      If distance d is less than or equal to radius r
         Increment counter by 1
   If counter is greater than or equal to m then
         This pixel on the raster surface is the center
         of a circle of radius r or smaller that
         contains at least m out of the n seeds, thus
         this is the center of an m-cluster
```

Fig. 2. Pseudo-code to find m-clusters on a raster surface

The complexity of this sequential algorithm is *O*(xSize*ySize*n). Gudmundsson's algorithm uses bounding squares instead of circles to define the distance (radius) test; and it is not equivalent to compare complexities with the algorithm presented here. It is clear that the complexity of the presented algorithm is high; however, it is not intended to be competitive with other approaches in a sequential setting. Its advantage lies in its suitability for highly efficient parallelism on a GPU.

2.2 Parallelized M-Clustering Algorithm

The pseudo-code presented in Figure 2 correctly calculates an approximation to the m-clusters but is inefficient when only a single processor is available. However, during the early 1990s, massively parallel machines such as the MasPar and the Connection Machine 2 allowed programmers to devise algorithms based on the assumption that a processor was available for every pixel [13]. As the processing power and the number of processing units inside a GPU increases, these types of algorithms are becoming relevant again. Fundamentally, a GPU's multiple processing units work in tandem in the same SIMD fashion as the earlier generation of parallel machines.

Flynn wrote: "If parallel processors are going to solve ... problems, the problems' representation should be designed for these machines. ... We need to represent problems in a cellular form, which could be addressed by computers in which each element has its own memory. This is crucial if we want to effectively use parallel processors." [14]

The sequential algorithm given in Figure 2 lends itself nicely to a rewrite in a programming language for GPUs, precisely in the spirit of Flynn's statement. Each processing element, which will be responsible for calculations of one pixel, need only contain enough local memory to hold the locations of all of the seeds. An added benefit is that the parallel version of the algorithm works with limited interprocess communications, as all calculations are local.

Since each processing element is responsible for one pixel, the computations performed in the sequential code for every element of the raster surface compose the CUDA kernel. The following pseudo-code sketches the CUDA implementation of the sequential algorithm presented above in Figure 2.

```
// In the Host
Allocate seeds in constant memory
Copy seeds to GPU
Allocate space (matrix) in GPU global memory
Clear matrix
Launch the kernel
Synchronize threads
Copy matrix back to Host
Free memory
// In the GPU (one pixel per processing element):
NumOfClosePoints = 0;
For each seed s
Calculate the distance from this pixel to seed s
      If (distance <= radius)
            NumOfClosePoints++;
If (NumOfClosePoints >= m)
Mark this pixel as the center of an m-cluster
```

Fig. 3. CUDA pseudo-code to find m-clusters on the raster surface

Notice that there are no data dependencies among instances of the kernel, implying that each instance of the kernel can execute independently of all others. The amount of memory declared and used inside each kernel is extremely small, which is important when programming today's GPUs with their limited number of internal registers.

Processing units inside GPUs are called stream processors. Current high-end GPUs contain 128 stream processors. CUDA's run-time environment manages scheduling and execution, in parallel, of as many instances of the kernel as the number of available stream processors. Thus, if there is one stream processor available for every pixel in the raster surface, then all kernels can execute simultaneously on the entire data set. Therefore, the time complexity of this parallel approximate algorithm is $O(\mathrm{n})$ where n is the number of seeds.

If the number of required instances of the kernel is larger than the number of processing units, i.e. the number of pixels in the problem is greater than the number of processing elements, then the run-time environment will schedule "batches" of kernel executions. This process continues until all required kernel instances have finished their execution.

Thus the parallel algorithm is independent of the number of available stream processors. If the number of pixels exceeds the number of stream processors, computation is automatically batched. If the number of available stream processors increases, the algorithm scales appropriately and correctly.

This algorithm is similar to a parallel algorithm for finding the Voronoi diagram and another for finding the Order-k Voronoi diagram as presented in [15]. The Voronoi diagram algorithm has recently been implemented using CUDA on an NVIDIA GeForce GPU and the complexity of the parallel algorithm has been shown to be linear on the number of seeds [16].

3 Finding Outliers

Statistically, an outlier is a point that is distant from the rest of the data and therefore does not belong. Using the definitions presented in section 2, we define an outlier as a point (a seed) that does not belong to any of the m-clusters found in the original set of points P. Correspondingly, it is possible to modify the cluster algorithm presented in the previous section to instead identify outliers.

The key observation is that as the algorithm calculates the center-to-seed distances required to identify an m-cluster, it is possible to record the identifications of the seeds that correspond to those distances. Thus, once a point in the raster surface is verified as the center of a circle of radius r or smaller that contains at least m seeds, the identities of each of those m seeds in the m-cluster are known. By definition, any seed not belonging to an m-cluster is an outlier.

The following example illustrates a set of seeds with one outlier. The raster surface is again assumed to be of size 12×12. There are five seeds with coordinates (1,1), (8,8), (8,10), (9,8) and (9,10). For this example, the maximum cluster radius r is defined to be 2.0 and each cluster must contain at least $m = 3$ seeds.

The state of the raster surface is depicted in Figure 4. In this diagram, the locations of the seeds are represented by the '+' symbol. The outlier seed, represented with an '*', is the seed located at coordinate (1,1).

Since our outlier algorithm is based on a modification of the m-clustering algorithm given in the previous section, we do not present any pseudo-code. A description of the required modifications follows.

In the sequential version of the outlier code, an array of counters, containing one entry for every seed, is initialized to zeroes. Whenever a point is identified as the

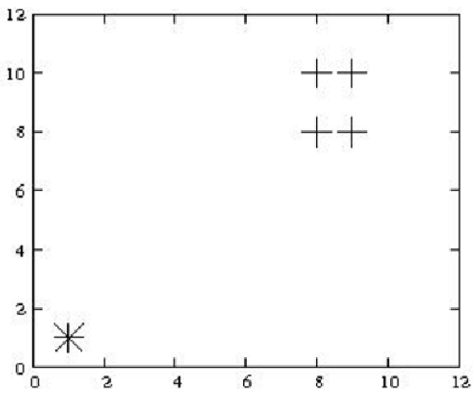


Fig. 4. The seed located at (1,1) is an outlier

center of an m-cluster, the counters for each of the seeds belonging to that m-cluster are incremented. Once the entire raster surface has been examined, the counters for the outlier seeds will still contain zeroes.

In the parallelized version of this code, each kernel maintains an array of integers called `localInCluster`. The array `localInCluster` contains one entry for every seed and all of entries are initialized to zero. A second array, referred to as `globalIn-Cluster`, is maintained in the shared memory area of the GPU. It also contains an entry for each seed, again initialized to zero. Each time the distance between a seed and the pixel associated with a kernel is determined to be within the radius *r*, the corresponding entry in `localInCluster` is set to 1. Thus, after calculating the distances from this pixel to all seeds, `localInCluster` contains a zero only in those entries corresponding to seeds that are at a distance greater than the radius from this pixel. If it is determined that the pixel associated with this kernel is the center of an m-cluster, all non-zero values in `localInCluster` are written to their corresponding entries in `globalIn-Cluster`. When all kernels have completed execution, the entries in `globalIn-Cluster` will contain zeroes only in the entries corresponding to seeds that were not part of any cluster. Those seeds are precisely the outliers.

The complexity of the parallel algorithm is again determined to be $O(n)$, where n is the number of seeds. Interestingly, the execution time of this algorithm was found to be very similar to that obtained for the m-clustering code. Unlike the data independent nature of the m-clustering code, the outlier code manifests the possibility of "write" contentions when updating the entries in the `globalInCluster` array. However, the similar execution times and speedups we obtained seem to indicate that contention is not a significant problem.

4 Results

To test the correctness of the parallel SIMD algorithms and the efficiency of the GPU implementations, the algorithms were executed on a machine with an NVIDIA GTS 8800 video card containing 128 stream processors.

There is no publicly available implementation of the Gudmundsson algorithm. However, there are commercial GIS programs that solve the same problem, which we use for comparison, to calculate the speedups obtained by our GPU implementation.

Several experiments were performed to test the scalability of our algorithms. Table 1 contains the execution times of the m-clustering algorithm for a set of 1024 seeds with increasing problem size, i.e. number of pixels. A superset of the data given in Table 1 is plotted in Figure 5.

Table 1. Execution time of the m-clustering algorithm for increasing problem size. Number of seeds is fixed at 1024.

Image size	Number of pixels	Execution time (ms)
256	65536	16.39
512	262144	43.35
1024	1048576	135.37
2048	4194304	473.14
4096	16777216	1703.89

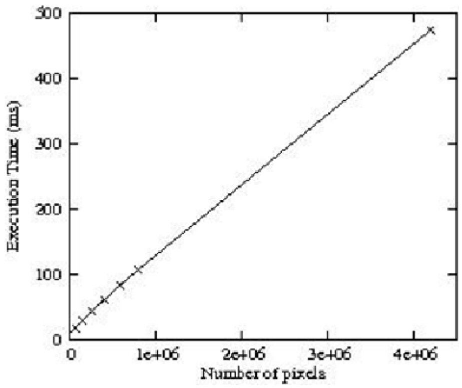


Fig. 5. Scalability of the m-clustering algorithm

The table and corresponding graph demonstrate that the algorithm scales well as the size of the image increases.

The next set of experiments tested whether the algorithm could scale linearly on the number of seeds. The size of the raster surface was fixed at 2048 × 2048, meaning the problem size was fixed at 4194304 pixels. As in the above experiment, the GTS 8800 video card was the test platform. Executions times are given below in Table 2 and plotted in Figure 6.

Once again it is apparent that the algorithm performs efficiently, scaling linearly as the number of seeds is increased. This suggests that this combination of data parallel algorithm and GPU technology will continue to be effective on ever larger datasets.

Table 2. Execution time of the m-clustering algorithm for increasing number of seeds. Problem size (number of pixels) is fixed at 4194304.

Number of seeds	Execution time (ms)
256	227.87
512	317.01
768	407.50
1024	498.71
1280	586.99
1536	677.40
1792	766.81
2048	856.44

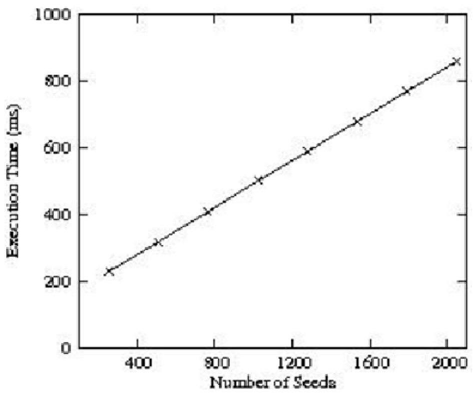


Fig. 6. Scalability of the m-clustering algorithm

Speedup tests were conducted against the commercial program ArcGIS v9.2 running on an Intel Core 2 duo 2.16GHz processor with 2GB RAM. The tests involved performing cluster and outlier analysis on the US Dam dataset. Given 1000 points (seeds), ArcGIS took 17 sec., while our algorithm took 0.499 sec. for 1024 points, a speedup over 34. To analyze 2000 points, ArcGIS took 36 sec., while our approach took 0.856 for 2048 points, a speedup over 42. Clearly, the sub-second performance of the parallelized method enables a more "interactive" or real-time experience for users. In addition, the large speedups suggest that it will be possible to perform data mining on the much larger datasets expected of the future.

5 Conclusion

We have presented two parallel raster algorithms optimized for GPUs: the first one finds clusters of at least *m* points in size, and the second one discovers outliers. Both algorithms have been implemented using CUDA, a programming language for GPUs. The performance results are encouraging and indicate that the algorithms are fast, scalable, linear in the number of seeds, and able to significantly outperform the implementation found in commercial programs.

The algorithms presented here take advantage of large numbers of stream processors and should scale well as the number of stream processors increases in newer GPUs. The low memory demand and the use of a raster format make these algorithms highly efficient in GIS spatial processing. Today's GPUs promise significant performance improvement for applications using raster processing in GIS systems.

References

1. Lo, C.P., Yeung, A.K.W.: Concepts and Techniques in Geographic Information Systems. Pearson Prentice Hall (2007)
2. Gudmundsson, J., van Kreveld, M., Narasimhan, G.: Region-Restricted Clustering for Geographic Data Mining. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 399–410. Springer, Heidelberg (2006)
3. Buck, I.: GPU computing with NVIDIA CUDA. In: Proceedings of 34th Conference on Computer Graphics (SIGGRAPH). ACM, New York (2007)
4. Christy, P.: Software to support massively parallel computing on the MasPar MP-1. In: Proceedings 35th IEEE Computer Society Conference (Compcon). IEEE, Los Alamitos (1990)
5. Rose, J.R., Steele, G.I.: C*: An extended C language for data parallel programming. In: Proceedings 2nd International Conference on Supercomputing (SC), pp. 2–16. ACM/IEEE (1988)
6. Braunl, T.: Parallaxis-III: Architecture-Independent Data Parallel Processing. Transactions on Software Engineering 26, 227–243 (2000)
7. Hoff, K.E., Culver, T., Keyser, J., Lin, M., Manocha, D.: Fast Computation of Generalized Voronoi Diagram and the Order-k Voronoi Diagram. In: Proceedings of 26th Conference on Computer Graphics (SIGGRAPH). ACM, New York (1999)
8. Fan, Q., Efrat, A., Koltun, V., Krishnan, S., Venkatasubramanian, S.: Hardware-Assisted Natural Neighbor Interpolation. In: Proceedings of 7th Workshop on Algorithm Engineering and Experiemtns (ALENEX). SIAM, Philadelphia (2005)
9. Denny, M.: Algorithmic Geometry via Graphics Hardware. PhD thesis, Universitaet des Saarlandes (2003)
10. Denny, M.: Solving Geometric Optimization Problems using Graphics Hardware. Computer Graphics Forum 22, 441–451 (2003)
11. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. ACM Computing Surveys 31, 264–323 (1999)
12. O'Sullivan, D., Unwin, D.J.: Geographic Information Analysis. John Wiley and Sons, Chichester (2003)
13. Blelloch, G.E., Little, J.J.: Parallel Solutions to Geometric Problems on the Scan Model of Computation. Tech. Report A.I. Memo 952, MIT Artificial Intelligence Lab (1988)
14. Flynn, M.J.: Parallel processors were the future...and yet be. Computer 29, 151–152 (1996)
15. Trefftz, C., Szakas, J.: Parallel Algorithms to find the Voronoi Diagram and the Order-k Voronoi Diagram. In: Proceedings 3rd Workshop on Massively Parallel Processing (IPDPS). IEEE, Los Alamitos (2003)
16. Majdandzic, I., Trefftz, C., Wolffe, G.: Computation of Voronoi Diagrams using a Graphics Processing Unit. In: Proceedings of 2008 IEEE Intl. Conf. on Electro/Information Technology (EIT). IEEE, Los Alamitos (2008)

Evaluating the Jaccard-Tanimoto Index on Multi-core Architectures

Vipin Sachdeva[1], Douglas M. Freimuth[2], and Chris Mueller[3]

[1] IBM Future Technologies Design Center, Indianapolis, IN
vsachde@us.ibm.com
[2] IBM Watson Research Center, Hawthorne, NY
dmfreim@us.ibm.com
[3] Pervasive Technologies Labs and University Information Technology Services, Indiana University, Bloomington, IN
chemuell@cs.indiana.edu

Abstract. The Jaccard/Tanimoto coefficient is an important workload, used in a large variety of problems including drug design fingerprinting, clustering analysis, similarity web searching and image segmentation. This paper evaluates the Jaccard coefficient on three platforms: the Cell Broadband Engine^{TM}processor Intel ®Xeon ®dual-core platform and Nvidia ®8800 GTX GPU. In our work, we have developed a **novel parallel algorithm specially suited for the Cell/B.E. architecture for all-to-all Jaccard comparisons, that minimizes DMA transfers and reuses data in the local store.** We show that our implementation on Cell/B.E. outperforms the implementations on comparable Intel platforms by 6-20X with full accuracy, and from 10-50X in reduced accuracy mode, depending on the size of the data, and by more than 60X compared to Nvidia 8800 GTX. In addition to performance, we also discuss in detail our efforts to optimize our workload on these architectures and explain how avenues for optimization on each architecture are very different and vary from one architecture to another for our workload. Our work shows that the algorithms or kernels employed for the Jaccard coefficient calculation are heavily dependent on the traits of the target hardware.

1 Introduction

Recent years have seen a resurgence in the number of hardware choices available to programmers. Multi-core processor architecture cores, which have multiple processing elements on a single chip are now the norm of the industry [1]. A vast number of hardware choices are now available to a high-performance computing programmer: general-purpose processors available from IBM, AMD and Intel have upto 8 cores, Cell/B.E. architecture has 8 special vector cores called SPEs and a PPC core called PPE, and more recently GPUs, capable of running hundreds of threads, primarily meant for graphics processing tasks are also being evaluated for high-performance computing. This has very important implications for many industries, which could now accelerate their workloads using these multi-core chips, and thus not relying only on single-thread performance. One such workload, which could benefit from multi-core technology is the **Jaccard-Tanimoto index** [2], which is is a correlation coefficient for determining the similarity between two binary strings, or bit vectors. Jaccard coefficient finds it's application in a wide variety of areas such as drug design [3], similarity searching on the Internet [4], financial applications [5] and social network analysis [6].

2 Jaccard Coefficient: Sequential and Parallel Algorithm

The Jaccard coefficient is mathematically defined as follows: given two equal length bit vectors x and y, with entries indexed from 0 to n, the Jaccard index computes:

$$Jaccard = \frac{c}{a+b+c}$$

Computationally for 2 vectors x and y, c is the number of bits set in the *and* product , and $a+b$ is the number of bits set in the *xor* product of the 2 vectors x and y.

Algorithm 1: Jaccard Index: All-to-all kernel computation

```
Data  : (1) $n$ vectors $S_1$, $S_2$, . . . . $S_n$ each of length $b$

Result: The Jaccard index $J(i,j)$ obtained by computing the $S_i$ vector
        with the $S_j$ vector, $i \leq n$, $j \leq n$.
begin
    $k = 0$
    (1) for ($i = 0; i \leq n; i = i + 1$) do
        (2) for ($j = i + 1; j \leq n; j = j + 1$) do
            (2.1) $J(k) = J(S_i, S_j)$
            $k = k + 1$

end
```

Algorithm 1 shows the overall Jaccard computation, in which every binary vector is being compared with every other vector; henceforth, we will refer to the computation in Algorithm 1 as the **Jaccard workload**. As can be seen, the **Jaccard workload** does not have any dependence among the values being stored at the different locations, since we can precompute the address of storing of any coefficient J_{ij} independently; the index for storing the (i,j) is $ni - \frac{i(i+1)}{2} - i - 1 + j$.

Algorithm 2 shows an optimal parallel load-balanced approach which assigns all the vector comparisons of n elements in a queue, and divides the queue elements equally among the processing elements p.

Algorithm 2: Parallel Jaccard Algorithm

```
Data  : (1) $n$ vectors $S_1$, $S_2$, . . . . $S_n$ each of length $b$
        (2) Threads having ID's 1, 2 .... $thread\_max$
        (3) An output array $J$ for storing the Jaccard coefficients
        (4) A queue $Q$ of length $\frac{n(n-1)}{2}$
Result: The Jaccard index $S(i,j)$ obtained by aligning the $S_i$ vector with
        the $S_j$ vector, $i \leq n$, $j \leq n$.
begin
    $k = 0$
    $queue\_length = \frac{n(n-1)}{2}$
    (1) for $(i = 0; i \leq n; i = i + 1)$ do
        (2) for $(j = i + 1; j \leq n; j + 1)$ do
            (2.1) $Q[k].first = S_i$
            (2.2) $Q[k].second = S_j$
            (2.3) $Q[k].first\_element = i$
            (2.4) $Q[k].second\_element = j$
            (2.5) $k++$

    (2) for $(i = thread\_id; i \leq queue\_length; i = i + thread\_max)$ do
        (2.1) $i = Q[i].first\_element$
        (2.2) $j = Q[i].second\_element$
        (2.3) $J[ni - \frac{i(i+1)}{2} - i - 1 + j] = J(Q[i].first, Q[i].second)$

end
```

3 Jaccard Index on Multi-core Architectures

3.1 Jaccard Index on the Cell/B.E. Architecture

The Jaccard-Tanimoto kernel for two vectors x and y, essentially consists of summing the bits in the *and* product and the *xor* product of the two vectors over their entire lengths, and dividing the bits set in the *and* product by this sum, in single precision. Mathematically,

$$Jaccard = \frac{count\_bits(x\ and\ y)}{count\_bits(x\ and\ y)\ +\ count\_bits(x\ xor\ y)}$$

where *count_bits* specifies the number of bits set to 1 in the vector. This kernel can be thus broken up into 4 primary operations:

– Computing the *and* (*and_xy*) and the *xor* (*xor_xy*) product of the 2 vectors along their entire lengths, where *and_xy* and *xor_xy* are the vectors denoting the *and* and the *xor* product of the two vectors.
– Counting the bits set in *and_xy* and *xor_xy*, to obtain *bits_and_xy* and *bits_xor_xy*.

– Summing *bits_and_xy* and *bits_xor_xy* to obtain *bits_sum_xy*.
– Divide *bits_and_xy* by *bits_sum_xy*, and typecast the value as *float* to obtain *jaccard_xy*.

SPEs have instruction support for 128-bit *and* and *xor* products, which can then be used for the first operation, defined previously. Step 2 is the *pop-count* operation used in several important kernels, along with Jaccard coefficient. There are several known algorithms for *pop-count* on general-purpose purpose processors, as we discuss in Section 4, but we make use of the *cntb* [7] instruction which is part of the SPE instruction set. The *cntb* counts the number of ones in a byte, which can then be used to count the bits in the *and* and the *xor* product of the two vectors. The *cntb* instruction applies to 128-bit values, and for a 128-bit number x will return 16 8-bit values in the 128-bit result, each 8-bit value denoting the number of ones in the 8-bits of the input vector. The vectors *bits_and_xy* and *bits_sum_xy* need to be then summed across their 128 bit lengths to count the total number of ones in the entire 128-bit vector.

Our current implementation for summing the bits uses *sumb* instruction across two vectors (in our case *bits_and_xy* and *bits_xor_xy*), with appropriate shuffling using the *si_shufb* instruction In case the length of the input vectors is more than 128 bits (in our case it is 256 bits), this process needs to be repeated in blocks of 128 bits each, until the entire input vector length is finished. Using this approach, we are able to compute two vectors of type *vector short*, the last elements of which are *total_bits_and_xy* and *total_bits_xor_xy* respectively. *cuflt* instruction, to change the *vector int* vectors to the *vector float* types. We extract the last elements of each of the sum vectors using the instruction *shli*, and then perform a scalar division, to return the result. This scalar division operation takes up more than 50% of the cost of the entire kernel in the SPE pipeline, excluding the DMA costs. We could also return the results to a lesser accuracy upto 12 bits, and then keep the entire kernel fully SIMDized: this could be implemented by computing the reciprocal estimate of the vector containing *total_bits_xor_xy*, and then multiplying it by the vector containing *total_bits_and_xy*, to find an approximated Jaccard coefficient. The advantage of this strategy is that the kernel is fully SIMDized, however at the expense of accuracy. . We show the results for both these approaches in the Section 6.

A Novel Parallel Jaccard Algorithm on Cell/B.E. architecture. Our parallel algorithm on Cell/B.E. for all-to-all comparisons finds a substantially optimal parallel solution through a runtime comparison of work allocated to every SPE.

Given n vectors numbered 0 to $n-1$, we divide the Jaccard workload as follows: The vectors are divided amongst the SPEs by allocating *nvecs* to each SPE in a round-robin fashion. A result vector of size *nvecs* floating point values is also allocated in the each of the SPU local store. When the first *nvecs* (we call them *master vectors*) are DMAed to the SPE local store, the SPE computes the Jaccard coefficient within each pair of the *nvecs* vectors. The result array for each i^{th} vector (comparisons from $(i+1)^{th}$ to $(nvecs-1)$ vectors) is stored to the

PPU memory, before the computation for vector $i+1$ begins. Each of the result values of (i, j) Jaccard comparison are actually 16-bytes to allow DMAs from any (i, j) value to the PPE memory. Before these all-to-all computations among *nvecs* vectors are completed, another DMA request for the next *nvecs* (*nvecs* to $2 * nvecs - 1$ indices) are sent: we call them *slave vectors*. Once the Jaccard computation among every pair in the *master vectors* is completed, the Jaccard coefficient among this new set of *nvecs* vectors (called *slave vectors*) and the master vectors is then initiated. This process of streaming in the *nvecs* vectors (*slave vectors* is repeated, until the entire set of n vectors in completed. Once the entire set of the slave vectors is completed, the next set of *master vectors* is streamed in; this process is continued until the end when all the computations are finished. Thus through this approach, we find the unique solution that maximizes reuse of the data streamed into the local store of the processing element. We perform all-to-all comparisons among every set of vectors that is streamed into the local store of the processing element, i.e. the master vectors (when they are streamed in), and then also between the master vectors and the slave vectors. This leads to maximal reuse of the data. In addition, our algorithm overlaps computation with communication to hide the latency of the data. For example, data is DMAed from main memory such that it arrives when comparisons among other data are being performed, e.g., when comparisons among master vectors are being performed, the slave vectors are being DMAed into the local store of the processing element. When one set of slave vector computations has started, the next set of slave vectors are already being streamed into the local store. In our experimental results, we actually vary the *nvecs* variable with the number of SPEs, to show the optimal number of *nvecs* variable for the number of SPEs. **Thus there are competing interests which are balanced by the runtime tests which select an *nvecs* value that is optimum based on these considerations.** We show the detailed results of the implementation so far in the Section 6, which show that we achieve a super-linear speedup by optimizing several characteristics.

4 Jaccard Coefficient on General-Purpose Processors

In the last few years, almost all the hardware manufactures including IBM, Intel and AMD have released multi-core general-purpose processors. For our comparisons, we decided to evaluate the Intel Xeon 5160 processor, which is a dual-core processor. Due to the availability of the scalar and vector registers (128 bits), with L1 and L2 caches for data access in general-purpose processors [8], there is a high number of algorithms, that we could experiment with for the entire Jaccard coefficient calculation and the pop-count operation, in general.

The fastest approach for the pop-count operation (as shown by runtime tests), which leverages the caches on the Xeon, is to precompute the pop-count of unsigned numbers upto x bits, and store them in a table. We could then simply perform a lookup of the table at the index (equal to the number itself), and find out directly the number of bits set to 1, for the pop-count. The number x has to be chosen, considering that a fast access to the table is extremely important for this strategy; the Xeon 5160 has an individual 32 KB of L1 cache for each core, and a shared 4 MB L2 cache shared between two cores. It is important that the table fits into the L2 cache atleast, since we are looking at random indices, we do not expect the prefetching strategy to work well; for our experiments, we chose the table to work for unsigned numbers upto 16-bits, or $(2^{16}-1)$ elements, which is 65535 numbers. Each of these numbers are stored as type *char* datatypes, which makes the total size of the table close to 65 KB, which fits the table easily into the L2 cache. Thus, for the popcount operation for 256-bit vectors as in our input, using the table lookup, we will have to do 16 table lookups each for the *and* and the *xor* vectors. We could use the table-lookup approach with the 128 bit *and* and the *xor* operations for the entire Jaccard coefficient. On the other hand, we could also use the general-purpose registers for the *and* and the *xor* operation as well. We could, thus do 64-bit *and* and *xor* operations, and then perform the pop-count using the table-lookup. It actually turns out that not using vector registers is infact faster, due to the overhead of moving data from vector registers to the scalar registers. We have manually unrolled the loops in using the 64-bit *and* and the *xor* operations, for optimal performance for the 256-bit vectors. Our implementation is parallelized by POSIX threads using Algorithm 2, and we report results for execution on multiple cores of the Xeon 5160.

5 Implementation of Jaccard Coefficient on GPUs

In recent years, there have been a number of efforts to use GPUs, as a high-performance computing platform for data-parallel computations. CUDA, recently released by Nvidia allows the programmers to use GPUs for scalable computing, without the need of mapping their problem in terms of OpenGL APIs. For further details on CUDA, please refer to [9] . On the Nvidia GPU, we have partitioned of the problem as explained in Algorithm 2, we now discuss some of the details specific to Nvidia platform. To begin the computation, the entire input array which we compute upon, is transferred to the GPU global memory using the CUDA APIs. The GPU we evaluated, 8800 GTX has 768 MB of global memory, which allows the output and the input arrays to be stored prior to the computation, for upto 8192 input vectors. The host CPU allocates memory on the GPU using *cudaMalloc* and transfers the input array to the GPU global memory using *cudaMemcpy* primitive which can be used to transfer memory to and from the GPU memory, based on the arguments given to the function. The computation is performed by the grid of thread blocks, with the number of threads in a block (Db) and the number of blocks in the grid (Dg), being specified as part of the function definition. These parameters are given in command-line by the user, to experiment with these values for optimal kernel execution. We allocate each of the input vectors, in a round-robin fashion among each thread as shown in Algorithm 2, which then proceeds to do the all-to-all computation with all the indices greater than the input vector index.

For our workload, we have declared a one-component *Db* and *Dg*, which refers to the number of threads in a block.

The *global* function, which is called by the host, to allocate and and transfer memory to the GPU memory, calls a *device* function, which actually runs the kernel of the Jaccard computation. We timed the various kernels on the GPU, to find out the fastest kernel executing on the GPU: our final kernel used is parallel bits approach, which uses the *right-shift*, *and* and the *addition* operation, for the pop-count operation: these operations are generally considered fast on the GPUs. We load the 256-bit input vectors using the built-in vector types *uint4*, since they allow 128-bit load instructions. Storing of the Jaccard coefficients as well as loading of the input vectors is still being performed to/from the global memory, without trying to use the shared memory resource for this. This, could be one of the bottlenecks in our implementation, as access to shared memory is much faster than access to the global memory. Due to space constraints, we are unable to give any further details on our implementation on GPUs.

6 Results

We show our results for three different multi-core architectures: Cell Broadband Engine heterogeneous chip, Intel Xeon 5160 and the Nvidia 8800 GTX. Our input datasets contain fingerprint bit vectors for 257271 of the compounds in the NCI compound database. The bit vectors are 166-bits long, zero padded to 256 bits. The bit vectors are stored as contiguous with 32 bytes per vector; thus the input dataset is more than 8 MB in size.

6.1 Experimental Setup

For all three platforms, we present results varying the number of input vectors and the number of threads on each platform. Our code uses a common set of C files, and are compiled by a GNU Makefile framework for compiling for all the three platforms. This common platform makes it easier, for us to work with a common code base. The Intel Xeon 5160 was a two-processor system, with two Intel Xeon 5160 dual-core processors running at 3.0 Ghz connected through a 1.33 Ghz system bus. Each dual-core has a 4 MB L2 cache, and a 32 KB L1 cache for data and a 32 KB L1 for instructions. Our platform for the Cell Broadband Engine uses the IBM BladeCenter QS21; QS21 has 2 Cell Broadband Engine chips running at 3.0 Ghz, that are also part of the Playstation 3. The GPU platform is Nvidia 8800 GTX, which has 16 multiprocessors, with the size of the global memory as 768 MB. The shared memory used among threads in a block is 16 KB, and the number of registers per multiprocessor is 8192. The host system is Intel Core 2 Quad CPU with more than 2 GB of memory, and runs Fedora Core 7 kernel 2.6.23. The host files were compiled with gcc version 4.1.2, and the CUDA files were compiled with nvcc version 1.1, the object files (CUDA and C) were linked in with gcc.

Table 1. Performance of Intel Xeon

Number of Input Vectors	**Threads**	**Time (in seconds)**
2048	1	49.76
	2	27.25
	4	21.13
4096	1	200.87
	2	105.59
	4	70.16
8192	1	832.21
	2	408.37
	4	257.87

6.2 Results and Analysis

Table 6.2 shows the results for the Intel Xeon 5160, upto 4 threads. The results represent the times for the fastest Jaccard kernel explained in Section 4; we time each of the kernels, and the time in the Table 6.2 represent the lowest time of all the available kernels. Each of the threads run on a single-core: thus, 2 threads are running on a single Intel Xeon 5160 chip, and 4 threads on both the chips. The parallelization has been implemented with POSIX threads.

Table 6.2 shows the results for the QS21 platform, with number of SPEs varying from 1 to 16; the times shown represent the best values by varying *nvecs* variable. We time execution on each SPE varying the *nvecs* variable, and the time in the Table 6.2 represent the lowest time for possible values of *nvecs*. The *nvecs* variable are varied from 16 to 1024 in our experiments, in multiples of 2. For lower SPEs, a higher *nvecs* gives the optimal result, as load-balancing is not of critical importance, but for higher SPEs, lower *nvecs* gives better results. Since each Cell Broadband Engine chip has 8 SPEs, 16 SPEs represents running the Jaccard coefficient on both the Cell Broadband Engine chips in the QS21. Comparing the results to the Intel platform, it is important we do a chip-to-chip comparison: **thus, results for 4 threads on the Intel platform is equivalent to results for 16 SPEs on the Cell Broadband Engine**. Table 6.2 shows the times for both the *full* and the *reduced* accuracy mode; more details on the reduced-accuracy mode were discussed in Section 3.1. **The fully SIMDized reduced-accuracy mode, actually is 2-4X faster than the one with the scalar division**. This is due to two factors: the scalar division on the SPE is actually being performed by multiple instructions, and thus ends up taking a significant time of the total kernel. Also, with the reduced-accuracy mode, we keep the total kernel completely SIMDized as well.

Table 6.2 shows the best results for the Jaccard workload on a single Nvidia 8800 GTX GPU. We time all possible combinations of threads and blocks; the times in Table 6.2 represent the lowest time for all possible combinations varying the thread layout (number of threads in a block *Db* and the number of blocks

Table 2. Performance of Cell/B.E. architecture with varying number of SPEs

Number of Input Vectors	**Number of SPEs**	**Time (in seconds) (reduced-accuracy)**	**Time (in seconds) (full-accuracy)**
2048	1	29.33	74.50
	2	16.37	40.25
	4	7.148	19.41
	8	1.481	7.96
	16	.242	0.815
4096	1	77.67	198.47
	2	42.846	107.61
	4	22.24	58.13
	8	10.58	27.13
	16	2.758	10.88
8192	1	320.814	801.18
	2	163.416	423.11
	4	89.15	219.66
	8	56.199	111.13
	16	23.12	42.85

Table 3. Performance of Nvidia 8800 GTX

Number of Input Vectors	**Time (reduced accuracy) in seconds**
2048	148.75
4096	587.87
8192	2385.14

in the grid *Dg*), in multiples of 2. The debugging switch mentioned before, will also compare the GPU execution results with the host machine to make sure the results are computed and stored correctly. As we mentioned before in Section 3.1, GPU division operation is not fully IEEE compliant, as it can only be performed through a reciprocal estimate. **Thus, these results are equivalent to the reduced-accuracy mode of the Cell Broadband Engine**. As can be seen from Table 6.2, GPU results so far are much slower than the Intel and the Cell/B.E. architecture; however it is to be noted that global memory is still being used for loading the inputs and storing of the results; this leads to reduced bandwidth and is a bottleneck in the implementation.

7 Conclusion

In this paper, we have evaluated the Jaccard workload on a variety of platforms including the Cell Broadband Engine, Nvidia 8800 GTX GPU and multi-core Intel Xeon 5160. **We have developed a novel parallel algorithm for the Cell Broadband Engine, that finds a substantially optimal parallel solution through a runtime comparison of work allocated to SPEs. The Cell Broadband Engine is shown to be upto 10X better in full accuracy, and upto 50X better in reduced accuracy mode over the comparable Intel platform.**

Acknowledgments

This research was supported in part by University Information Technology Services and the Pervasive Technology Labs of Indiana University. Pervasive Technology Labs of Indiana University is in part funded by the Lilly Endowment, Inc. We wish to thank IBM Austin Research Laboratory for access to the Intel Xeon 5160 dual-core and the Nvidia 8800 GTX platforms. We also wish to thank Poughkeepsie Benchmarking Center for access to the QS21 platform.

References

1. Geer, D.: Industry trends: Chip makers turn to multicore processors. IEEE Computer 38(5), 11–13 (2005)
2. Jaccard, P.: The distribution of flora in the alpine zone. New Phytologist 11, 37–50 (1912)
3. Willett, P.: Similarity-based virtual screening using 2D fingerprints. Drug Discovery Today 11, 1046–1053 (2006)
4. Murata, T.: Machine discovery based on the co-occurence of references in search engine. In: Arikawa, S., Furukawa, K. (eds.) DS 1999. LNCS, vol. 1721, pp. 220–229. Springer, Heidelberg (1999)
5. Mild, A., Reutterer, T.: Proc. Sixth Int'l. Computer Science Conf. on Active Media Technology, Hong Kong, China, December 2001, pp. 302–313 (2001)
6. Liben-Nowell, D., Kleinberg, J.: The link prediction problem for social networks. In: Proc. of the Twelfth Int'l. Conf. on Active Media Technology, New Orleans, LA, pp. 302–313 (2003)
7. IBM: SPU instruction set architecture (2007), `http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/76CA6C7304210F3987257060006F2C44/file/SPU_ISA_v1.2_27Jan2007_pub.pdf`
8. El-Qawasmeh, E.: Performance investigation of bit-counting algorithms with a speedup to lookup table. Journal of Research and Practice in Information Technology 32(3), 215–230 (2000)
9. Nvidia: CUDA programming guide 1.1. (November 2007), `http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf`

Pairwise Distance Matrix Computation for Multiple Sequence Alignment on the Cell Broadband Engine

Adrianto Wirawan, Bertil Schmidt, and Chee Keong Kwoh

School of Computer Engineering, Nanyang Technological University, Singapore 639798
{adri0004,asbschmidt,asckkwoh}@ntu.edu.sg

Abstract. Multiple sequence alignment is an important tool in bioinformatics. Although efficient heuristic algorithms exist for this problem, the exponential growth of biological data demands an even higher throughput. The recent emergence of accelerator technologies has made it possible to achieve a highly improved execution time for many bioinformatics applications compared to general-purpose platforms. In this paper, we demonstrate how the PlayStation®3, powered by the Cell Broadband Engine, can be used as a computational platform to accelerate the distance matrix computation utilized in multiple sequence alignment algorithms.

Keywords: multiple sequence alignment, cell broadband engine.

1 Introduction

Multiple sequence alignment (*MSA*) of many nucleotides or amino acids is an important tool in bioinformatics. It can identify patterns or motifs to characterize protein families, and is therefore utilized to detect homology between sequences as well as to perform phylogenetic analysis. Many MSA heuristics have been proposed to reduce the exponential complexity of computing optimal MSAs. Heuristic MSA implementations include MSA[1], ClustalW[2], T-Coffee[3], MAFFT[4], DIALIGN P[5] and PRALINE[6]. ClustalW[2] has over 26,000 citations in the ISI Web of Science and is considered to be one of the most popular MSA tools. It is based on the progressive alignment method. Although not optimal, this method can produce reasonably good alignments at a good efficiency. However, the exponential growth of biological data demands an even better throughput. Thus, software approaches to improve the performance of ClustalW have been introduced, including caching[8, 9] and parallel processing[10-12].

The recent emergence of accelerator technologies such as FPGAs, GPUs and specialized processors have made it possible to achieve an improvement in execution time for many bioinformatics applications compared to current general-purpose platforms at a low cost. Recent usage of easily accessible accelerator technologies to improve the ClustalW algorithm include FPGA[13] and GPU[14].

Our profiling of ClustalW has revealed that *distance matrix computation* is the most time consuming stage and typically takes up more than 90% of the overall runtime.

Therefore, accelerating this phase would greatly improve the performance as a whole. In this paper, we investigate how the Cell Broadband Engine, a heterogeneous multi-core architecture, can be used as a computational platform to accelerate the distance matrix computation in ClustalW. By taking advantage of multiple processors as well as SIMD vectorization, we are able to achieve speedups of two orders of magnitude compared to the publicly available sequential ClustalW implementation.

The rest of this paper is organized as follows. Section 2 highlights important features of the Cell Broadband Engine. Section 3 briefly describes the ClustalW algorithm. Section 4 presents our mapping of the distance matrix computation onto the Cell Broadband Engine. Experimental results are presented in Section 5. Section 6 concludes the paper.

2 Cell Broadband Engine

The Cell Broadband Engine[15] (*Cell BE*) is a recently introduced single-chip heterogeneous multi-core processor, which is developed by Sony, Toshiba and IBM. The Cell BE offers a unique assembly of thread-level and data-level parallelization options. It is operating at the upper range of existing processor frequencies (3.2 GHz for current models). Several examples of bioinformatics applications that has been ported to the Cell BE architecture include Folding@Home[16], FASTA[17], Smith-Waterman[18] and RAxML[19].

The Cell BE combines an IBM PowerPC Processor Element (PPE) and eight Synergistic Processor Elements (SPEs)[20]. An integrated high-bandwidth bus called the Element Interconnect Bus (EIB) connects the processors and their ports to external memory and I/O devices. The block diagram of the Cell BE architecture is shown in Figure 1.

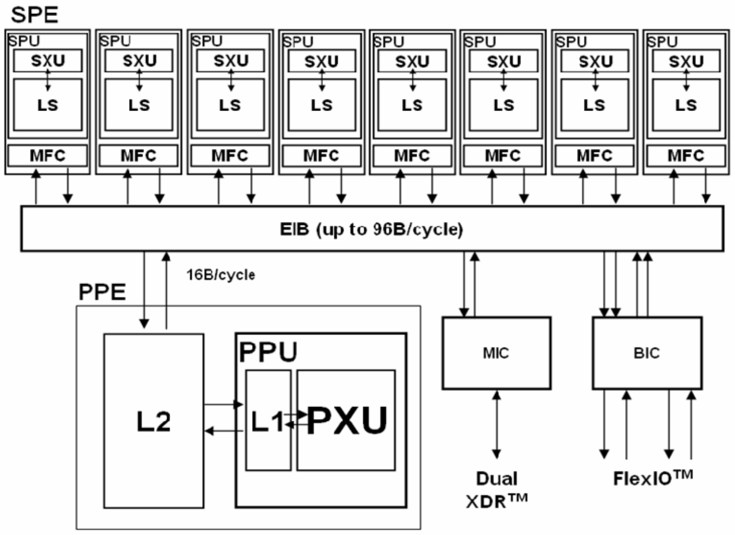


Fig. 1. Block diagram of the Cell Broadband Engine Architecture

The PPE is a 64-bit Power Architecture core and contains a 64-bit general purpose register set (GPR), a 64-bit floating point register set (FPR), and a 128-bit Altivec register set. It is fully compliant with the 64-bit Power Architecture specification and can run 32-bit and 64-bit operating systems and applications. Each SPE is able to run its own individual application programs. Each SPE consists of a processor designed for streaming workloads, a local memory, and a globally coherent Direct Memory Access (DMA) engine. The EIB is a 4-ring structure, and can transmit 96 bytes per cycle, for a bandwidth of 204.8 Gigabytes/second. The EIB can support more than 100 outstanding DMA requests.

The most distinguishing feature of the Cell BE lies within the variety of the processors it has, i.e. the PPE and the SPEs. Heterogeneous multi-core systems can lead to decreased performance if both the operating system and application are unaware of the heterogeneity. The PPE is designed to run the operating system and, in many cases, the top-level control thread of an application, while the SPEs is optimized for compute intensive applications, hence, providing the bulk of the application performance.

The SPE can access RAM through direct memory access (DMA) requests. The DMA transfers are handled by the Memory Flow Controller (MFC). The MFC provides the interface, by means of the EIB, between the local storage of the SPE and main memory. The MFC supports DMA transfers as well as mailbox and signal-notification messaging between the SPE and the PPE and other devices. Data transferred between local storage and main memory must be 128-bit aligned. The size of each DMA transfer can be at most 16 KB. DMA-lists can be used for transferring large amounts of data (more than 16 KB). A list can have up to 2,048 DMA requests, each for up to 16 KB.

The PS3 uses the Cell Broadband Engine as its CPU, hence making it possible for users to create a high-powered computing environment for a fraction of the cost of a Cell Blade server. The PS3 utilizes seven of the eight SPEs, in which the eighth SPE is disabled to improve chip yields, i.e. chips do not have to be discarded if one of the SPEs is defective. Only six of the seven SPEs are accessible to developers as one is reserved by the operating system. Generally available PS3's can be used for scientific high performance computing through installation of Linux (e.g. Fedora Core or Yellow Dog). Programs can be developed the using freely available C-based Cell BE SDK[21].

3 Multiple Sequence Alignment

ClustalW is highly popular sequential MSA software. It implements a progressive alignment method[22], i.e. it adds sequences one by one to the existing alignment to build a new alignment. The order of sequences to be added to the new alignment is indicated by a phylogenetic tree, which is called a guide tree. The guide tree is constructed from the similarity of all possible pairs of sequences stored in the distance matrix. Overall, the three stages of the ClustalW algorithm can be summarized as follows:

1. *Distance matrix computation*: each pair of sequences is aligned separately to calculate a respective distance value. These values are stored in a so-called distance matrix.
2. *Guide tree construction*: the guide tree is calculated from the distance matrix using the neighbor-joining algorithm[23]. The guide tree defines the order which the sequences are aligned in the next stage.
3. *Progressive alignment*: The sequences are progressively aligned in accordance to the guide tree.

Given n number of sequences of length m, the distance matrix computation has a quadratic complexity of $O(n^2m^2)$. Profiling the three stages of ClustalW using *gprof* shows that the distance matrix computation is the most computationally intensive phase and typically takes up more than 90% of the overall runtime. Hence, it can be concluded that accelerating the distance matrix computation would provide a good speed up for the ClustalW.

Given a set of n sequences $S = \{S_1, S_2, \ldots, S_n\}$. The distance of two sequences S_i, $S_j \in S$, is defined by Equation (1).

$$d(S_i, S_j) = 1 - \frac{nid(S, S_j)}{\min\{l_i, l_j\}} \tag{1}$$

where $nid(S_i,S_j)$ denotes the number of exact matches in the *optimal local alignment* of S_i and S_j with respect to the given scoring system and l_i and l_j denotes the length of S_i and S_j, respectively.

Liu et.al[14] states that given two sequences S_1 and S_2 with affine gap penalties α and β and the substitution table *sbt*, a matrix $N_A(i,j)$ $(1 \leq i \leq l_1,\ 1 \leq j \leq l_2)$ can be recursively defined as shown in Equation 2.

$$N_A(i,j) = \begin{cases} 0, & \text{if } H_A(i,j) = 0 \\ N_A(i-1,j-1) + m(i,j), & \text{if } H_A(i,j) = H_A(i-1,j-1) + sbt(S_1[i], S_2[j]) \\ N_E(i,j-1), & \text{if } H_A(i,j) = E(i,j) \\ N_F(i,j), & \text{if } H_A(i,j) = F(i,j) \end{cases} \tag{2}$$

where

$$m(i,j) = \begin{cases} 1, \text{if } S_1[i] = S_2[j] \\ 0, \text{otherwise} \end{cases}$$

$$N_E(i,j) = \begin{cases} 0, & \text{if } j = 1 \\ N_A(i,j-1), \text{if } E(i,j) = H_A(i,j-1) - \alpha \\ N_E(i,j-1), & \text{if } E(i,j) = E(i,j-1) - \beta \end{cases}$$

$$N_F(i,j) = \begin{cases} 0, & if\ j = 1 \\ N_A(i-1,j), & if\ F(i,j) = H_A(i-1,j) - \alpha \\ N_F(i-1,j), & if\ F(i,j) = F(i-1,j) - \beta \end{cases}$$

$$H_A = \max \begin{cases} 0, \\ E(i,j), \\ F(i,j), \\ H_A(i-1,j-1) + sbt(S_1[i], S_2[j]) \end{cases}$$

$$E(i,j) = \max\{H_A(i,j-1) - \alpha, E(i,j-1) - \beta\}$$

$$F(i,j) = \max\{H_A(i-1,j) - \alpha, F(i-1,j) - \beta\}$$

Using the matrix $N_A(i,j)$, the distance value $d(S_i,S_j)$ can then be redefined as shown in Equation 3.

$$d(S_i,S_j) = 1 - \frac{NA(i_{\max}, j_{\max})}{\min\{l_i, l_j\}} \tag{3}$$

A more detailed explanation and proof of these formulas is described in[14].

4 Mapping onto the Cell BE Architecture

Figure 2 illustrates the mapping of distance matrix computation onto the Cell BE.

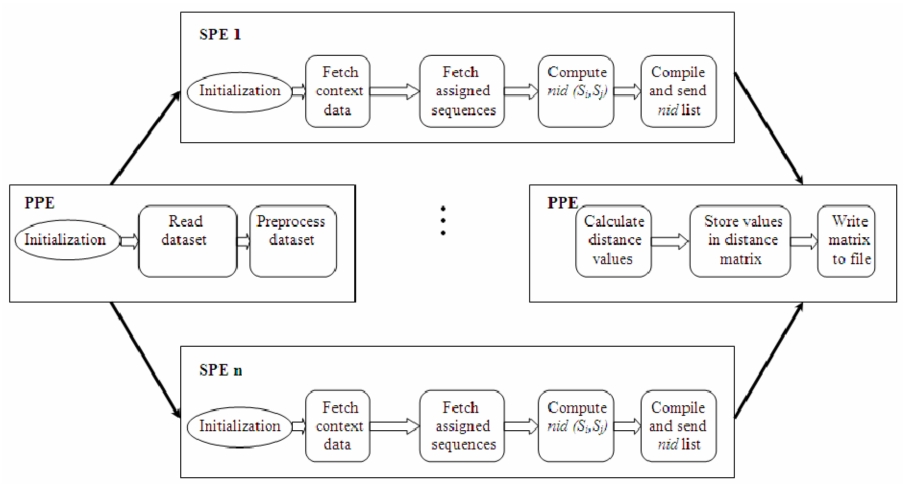


Fig. 2. Mapping of our distance matrix computation algorithm onto the Cell B.E.

The algorithm starts by reading the input dataset. The PPE then preprocesses and divides the dataset into equal size blocks for each SPEs to process. Since the blocks are independent of each other, no thread synchronization is necessary during the calculations. The mailbox functions *spe_in_mbox_write* and *spu_read_in_mbox* are used to ensure that all the SPEs obtain their respective contexts in their local memory. Using the context data, each SPE then transfers any required information and necessary sequences. To improve transfer efficiency, the database sequences in main memory and in the local storage are aligned within the cache line and data structures are initialized during the transfer of the sequence. Once it has finished calculating all its respective $nid(S_i,S_j)$ scores, each SPEs sends the scores to the PPE in the form of a list. The PPE compiles the lists, calculates the distance values, and stores them in the distance matrix. The matrix is then written to a text file. The SPE pseudocode is shown in Figure 3.

```
Initialization;
Fetch the context data from the mailbox;
Fetch the set of sequences using DMA transfer;
While there are sequences to be processed
    Calculate nid score;
    Compile the nid scores into a list nidlist;
Send nidlist to PPE using DMA transfer;
```

Fig. 3. Pseudocode of the SPE code

The $nid(S_i,S_j)$ scores are computed in the SPEs in parallel using the Single Instruction Multiple Data (SIMD) registers using SPU intrinsics[24]. The pseudocode of the *nid* score calculation is shown in Figure 4.

Based on Equation (3), $nid(S_i,S_j)$ can be computed without computation of the actual traceback, which can cost a lot of resources from storing the complete dynamic programming matrix especially for long sequences. Since all elements in the same minor diagonal of the dynamic programming matrix can be computed independent of each other in parallel, the computation is done in minor diagonal order. Given are the vector registers vH, vE, vF, vN_A, vN_E and vN_F containing the values H_A, E, F, N_A, N_E and N_F, respectively. For each loop of c ($1 \le c \le (l_1+l_2-1)$), the values of H_A, E, F, N_A, N_E and N_F are computed. Calculations of the vH, vE, and vF vectors are done by utilizing the *spu_cmpgt* intrinsic, which compares each element of a vector with the corresponding element of another vector, to create vector masks. The masks are then used as patterns to generate the resulting vector using the *spu_sel* intrinsic, which selects the corresponding bit from either vector in accordance to a provided pattern vector. The masks used in the vE, vF and vH computations are used to determine the value of the corresponding vN_E, vN_F and vN_A vectors, respectively.

```
Initialization;
Load gOpen to vector vGapOpen;
Load gExtend to vector vGapExtend;
For $c$ = 1 to $l_1+l_2-1$
    Load the necessary vector registers for
        minor diagonal computations;
    Calculate vector register of E vE;
    Calculate vector register of $N_E$ $vN_E$;
    Calculate vector register of F vF;
    Calculate vector register of $N_F$ $vN_F$;
    Calculate vector register of $H_A$ vH;
    Calculate vector register of $N_A$ $vN_A$;
Extract nid as $N_A(i_{max},j_{max})$;
Return nid;
```

Fig. 4. Pseudocode of nid score calculation

5 Performance Evaluation

A set of experiments has been conducted using different numbers of protein sequences i.e. 400 sequences of average length 408, 600 sequences of average length 462, 800 sequences of average length 454, and 1000 sequences of average length 446. The measured runtimes are then compared to the original ClustalW implementation and the GPU-ClustalW implementation described in[14].

All experiments have been carried out on a standalone PS®3 with Fedora Core 9.0 operating system and the Cell Software Development Kit (SDK) 3.1. The sequential ClustalW application, available online at http://www.bii.a-star.edu.sg/achievements/applications/clustalw/, was benchmarked on an Intel Pentium 4 3.0 GHz processor with 1 GB RAM running on Windows XP. The GPU-ClustalW implementation was conducted with Nvidia GeForce 7900 GTX graphic card with a 717 MHz engine clock speed, a 1.79 GHz memory clock speed, 8 vertex processors, 24 fragment processors, and a 512 MB memory, running on Pentium 4 3.0 GHz, 1 GB RAM with Windows XP.

Table 1 shows the performance evaluation of our parallel algorithm using the above mentioned datasets. The term *n*(*m*) describes a dataset containing *n* sequences

Table 1. Runtime comparison of distance matrix computation. The timing is measured in seconds.

#sequences (average length)	400 (408)	600 (462)	800 (454)	1000 (446)
ClustalW	833.1	1697.0	2966.6	4409.6
GPU-ClustalW	73.7	150.0	249.0	368.8
Playstation® 3	11.0	20.4	29.5	40.8

with an average length of *m*. By using all 6 SPEs available on the Playstation®3, our parallel algorithm achieves a runtime of 40.82 seconds for a dataset input of 1000 sequences of average length 446.

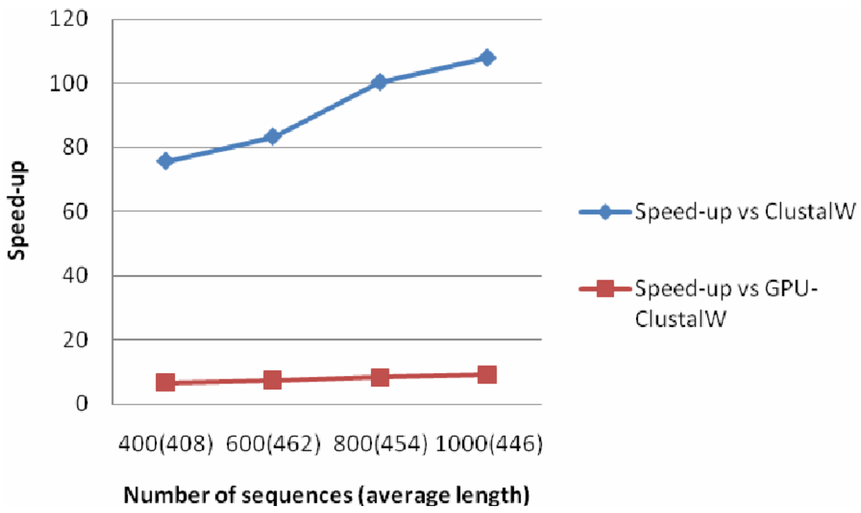


Fig. 5. Speed-up comparison of our implementation on the Playstation® 3 with ClustalW and GPU-ClustalW

Figure 5 shows the speed-up obtained by our implementation compared to the ClustalW and GPU-ClustalW. Our implementation obtains an average speed-up of 91.87x over all the datasets compared to the ClustalW implementation, with a peak speed-up of 108.03x for the 1000(446) dataset. The average speed-up of our implementation over the GPU-ClustalW is 7.87x, with a peak speed up of 9.03x for the 1000(446) dataset.

Table 2. Detailed performance analysis of our parallel algorithm. The terms *T* and *S* describe the runtime and the speed-up compared to the previous row, respectively.

#sequences (average length)	Processor	400 (408)		600 (462)		800 (454)		1000 (446)	
		T	S	T	S	T	S	T	S
Baseline ClustalW	Pentium 4 3.0 GHz	833.1	N.A	1697.0	N.A	2966.6	N.A	4409.6	N.A
Baseline ClustalW	PPE	667.86	1.24	1361.13	1.24	2379.0	1.24	3536.2	1.24
Non-vectorized code	PPE	357.89	1.87	717.83	1.89	1702.08	1.80	1871.08	1.89
Vectorized code	PPE+1SPE	57.15	6.26	113.41	6.33	168.54	7.83	237.12	7.89
Vectorized code	PPE+6SPEs	11.01	5.19	20.36	5.57	29.53	5.71	40.82	5.81

Table 2 shows a more detailed performance analysis of our parallel algorithm using the above mentioned datasets. It compares the runtimes of our implementation and the baseline ClustalW on various processors. The performance analysis breaks down the speedup obtained by each phase of the improvement made by our implementation. The non-vectorized code is implemented according to the algorithm described in section 3, without the use of SIMD vectorization. The vectorized code is implemented according to section 4.

6 Conclusion

We have presented a parallel algorithm on the Cell B.E. heterogeneous multi-core system for the distance matrix computation used in multiple sequence alignment algorithms. Our implementation on the Playstation®3 achieves an average speed-up of 91.87x compared to the publicly available sequential ClustalW implementation.

References

1. Lipman, D.J., Altschul, S.F., Kececioglu, J.D.: A tool for multiple sequence alignment. Proceedings of the National Academy of Sciences of the United States of America 86(12), 4412–4415 (1989)
2. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucl. Acids Res. 22(22), 4673–4680 (1994)
3. Notredame, C., Higgins, D.G., Heringa, J.: T-coffee: A novel method for fast and accurate multiple sequence alignment. Journal of Molecular Biology 302(1), 205–217 (2000)
4. Katoh, K., Misawa, K., Kuma, K.I., Miyata, T.: MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. Nucleic Acids Research 30(14), 3059–3066 (2002)
5. Schmollinger, M., Nieselt, K., Kaufmann, M., Morgenstern, B.: DIALIGN P: Fast pair-wise and multiple sequence alignment using parallel processors. BMC Bioinformatics, 5 (2004)
6. Simossis, V.A., Heringa, J.: PRALINE: A multiple sequence alignment toolbox that integrates homology-extended and secondary structure information. Nucleic Acids Research 33(suppl. 2), W289–W294 (2005)
7. Thompson, J.D., Gibson, T.J., Plewniak, F., Jeanmougin, F., Higgins, D.G.: The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. Nucl. Acids Res. 25(24), 4876–4882 (1997)
8. Catalyurek, U., Stahlberg, E., Ferreira, R., Saltzt, J.: Improving Performance of Multiple Sequence Alignment Analysis in Multi-client Environments. In: Proceedings of the First International Workshop on High Performance Computational Biology 2002 (HiCOMB 2002, IPDPS 2002) (2002)
9. Catalyurek, U., Gray, M., Kurc, T., Saltzt, J., Stahlberg, E., Ferreira, R.: A component-based implementation of multiple sequence alignment. In: Proceedings of the ACM Symposium on Applied Computing: 2003, pp. 122–126 (2003)
10. Li, K.-B.: ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics 19(12), 1585–1586 (2003)

11. Chaichoompu, K., Kittitornkun, S., Tongsima, S.: MT-ClustalW: Multithreading multiple sequence alignment. In: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006 (2006)
12. Luo, J., Ahmad, I., Ahmed, M., Paul, R.: Parallel multiple sequence alignment with dynamic scheduling. In: International Conference on Information Technology: Coding and Computing, ITCC 2005, pp. 8–13 (2005)
13. Oliver, T., Schmidt, B., Nathan, D., Clemens, R., Maskell, D.: Multiple sequence alignment on an FPGA. In: Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS 2005, pp. 326–330 (2005)
14. Liu, W., Schmidt, B., Voss, G., Muller-Wittig, W.: Streaming Algorithms for Biological Sequence Alignment on GPUs. IEEE Transactions on Parallel and Distributed Systems (2007)
15. Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R., Shippy, D.: Introduction to the Cell multiprocessor. IBM Journal of Research and Development 49(4-5), 589–604 (2005)
16. Pande, V.: Folding@Home: Using Worldwide distributed computing to break fundamental barriers in molecular simulation. In: Proceedings of the IEEE International Symposium on High Performance Distributed Computing 2006, p. 4 (2006)
17. Sachdeva, V., Kistler, M., Speight, E., Tzeng, T.-H.K.: Exploring the viability of the Cell Broadband Engine for bioinformatics applications. In: IEEE International Parallel and Distributed Processing Symposium 2007, 8 p. IEEE, Long Beach (2007)
18. Wirawan, A., Kwoh, C.K., Hieu, N.T., Schmidt, B.: CBESW: Sequence alignment on the playstation 3. BMC Bioinformatics, 9 (2008)
19. Stamatakis, A., Ludwig, T., Meier, H.: RAxML-II: A program for sequential, parallel and distributed inference of large phylogenetic trees. Concurrency Computation Practice and Experience 17(14), 1705–1723 (2005)
20. Pham, D., Behnen, E., Bolliger, M., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Le, B., Masubuchi, Y., et al.: The design methodology and implementation of a first-generation CELL processor: a multi-core SoC. In: Proceedings of the IEEE 2005 Custom Integrated Circuits Conference 2005, San Jose, CA, USA, pp. 45–49. IEEE, Los Alamitos (2005)
21. International Business Machines: Software Development Kit 2.1 Accelerated Library Framework Programmer's Guide and API Reference, Version 1.1. In: IBM developerWorks (2007)
22. Feng, D.F., Doolittle, R.F.: Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. Journal of Molecular Evolution 25(4), 351–360 (1987)
23. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular biology and evolution 4(4), 406–425 (1987)
24. IBM: C/C++ Language Extensions for Cell Broadband Engine Architecture v.2.5. In. IBM developerWorks (2008)

Evaluation of the SUN UltraSparc T2+ Processor for Computational Science

Martin Sandrieser, Sabri Pllana, and Siegfried Benkner

University of Vienna, Department of Scientific Computing,
Nordbergstrasse 15, 1090 Vienna, Austria
{ms,pllana,sigi}@par.univie.ac.at

Abstract. The Sun UltraSparc T2+ processor was designed for throughput computing and thread level parallelism. In this paper we evaluate its suitability for computational science. A set of benchmarks representing typical building blocks of scientific applications and a real-world hybrid MPI/OpenMP code for ocean simulation are used for performance evaluation. Additionally we apply micro benchmarks to evaluate the performance of certain components (such as the memory subsystem). To recognise the capabilities of the T2+ processor we compare its performance with the IBM POWER6 processor. While the UltraSparc T2+ is targeted on server workloads with high throughput requirements via low-frequency core design and massive chip multithreading capabilities, the ultra-high frequency core design of the IBM POWER6 optimised for instruction-level parallelism follows a contrary approach. The intention of this evaluation is to investigate whether the current generation of massive chip multithreading processors is capable of providing competitive performance for non-server workloads in scientific applications.

Keywords: Sun UltraSparc T2+, Niagara2, Evaluation, Computational Science.

1 Introduction

The improvement of single-core processor performance has reached technological limits known as power, memory, and instruction-level parallelism (ILP) walls. Chip multithreading technologies (CMT) [15] are considered as a viable approach to overcome most of these limitations. CMT is based on the employment of multiple cores per chip usually in combination with simultaneous multithreading (SMT) capabilities. CMT designs adhere to the principle that with lower frequencies and a higher number of SMT enabled cores, performance gains can be achieved while keeping power consumption at modest levels. These CMT/SMT design principles are incorporated in the Sun UltraSparc T2 processor (codenamed Niagara-2) [14].

The Sun UltraSparc T2 was designed as a scalable solution for fast growing datacenter applications, its main target being emerging web- and database markets. The massively multithreaded nature of the T2 chip reflects the market's need for a high throughput solution that can serve hundreds of clients on one single processor die. In April 2008 the T2+ processor [18] was introduced as an SMP extended version of the T2 processor allowing multiple processors to be used within a single system. The achievable savings in system costs per customer make massively multithreaded architectures very appealing to growing businesses, but for computationally intensive scientific computing applications it is unclear if these architectures can provide enough raw computational power.

In this paper we evaluate the suitability of the T2+ processor for computational science. A comprehensive measurement-based performance analysis study of the T2+ processor involves microbenchmarks, a collection of typical scientific kernels, and a real world ocean simulation application. We use microbenchmarks for the performance evaluation of certain system components. This includes an evaluation of the sustainable memory performance with the STREAM memory microbenchmark [8]. The NAS parallel benchmarks [1] serve as representatives of commonly used building blocks for scientific applications. To complement our performance evaluation study we use a real-world hybrid MPI/OpenMP Fortran90 code [4] that simulates the western intensification of wind-driven ocean currents.

To highlight the limitations and capabilities of the T2+ processor we provide a performance comparison with the IBM POWER6 [6] processor. While the UltraSparc T2+ is designed for server workloads with high throughput requirements via low-frequency core design and massive chip multithreading capabilities, the ultra-high frequency core design of the IBM POWER6 optimised for instruction-level parallelism follows a contrary architectural approach. Our aim is to find out if the current generation of massive chip multithreading processors is able to provide competitive performance for non-server workloads in scientific applications.

This paper is structured as follows. Section 2 gives an overview of the tested computing systems. The benchmarks are presented in Section 3. Section 4 presents and discusses the results of our performance analysis. Related work is discussed in Section 5. Finally, Section 6 concludes the paper and briefly describes future work.

2 Experimentation Platforms

In this section we describe the hardware and software environments of the computer systems used in our performance analysis study: (1) Sun T5140 Server and (2) IBM BladeCenter JS22. Table 1 summarises the main features of the analysed systems.

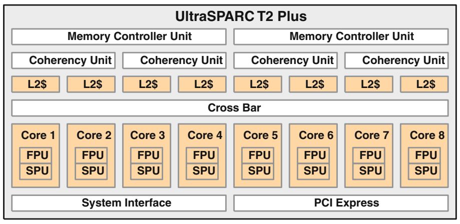


Fig. 1. T2+ processor block diagram[18]

SUN UltraSparc T2 Plus. The Sun T5140 Server [17] is comprised of two Sun UltraSPARC T2 Plus (codenamed Niagara-2) multi-core processors. The T2 Plus processor [14], which was introduced in April 2008, is an SMP extended version of the T2 processor allowing multiple Chip-level MultiThreading (CMT) processors to be used within a single system (shown in Figure 1). In the T2 architecture, massive chip multithreading is favoured over single thread performance. With concurrent execution possibilities of 64 threads per chip sustained processor performance and scalability are crucially dependent on efficient memory access mechanisms. Hence, the eight SPARC cores are

Table 1. Testing environments

System	Sun T5140 Server	IBM BladeCenter JS22
CPU's	2x 1.2 GHz Sun UltraSparc T2+	2x 4 GHz POWER6
Cores (System)	8(16)	2(4)
L1 instr. cache (core)	16KB	64KB
L1 data cache (core)	8KB	64KB
L2 cache (CPU)	4MB	2x 4MB
L3 cache (CPU)	-	32MB
HW Threads (CPU)	8x 8 (SMT)	2x 2 (SMT)
HW Threads (System)	128	8
Memory	8x 4GB 667MHz FBDIMM	8GB DDR2 DRAM
OS	Solaris 10 5/08	AIX 6.1
C Compiler	SUN Studio C 5.9	IBM XLC/C++ 10.1
Fortran Compiler	Sun Fortran 95 8.3	IBM XLF 12.1
Optimization	-O3 (max)	-O5 (max)
MPI	OpenMPI 1.2.5	-

connected via a high bandwidth crossbar to eight memory banks of a shared 4MB L2 cache. Each core can execute up to eight threads simultaneously. One core provides two integer execution units (EXU), one floating point and graphics unit (FPU) and a specialised stream processing unit (SPU) for cryptographic acceleration. The L2 cache banks are connected to two memory controllers (MCU).

IBM POWER6. The IBM BladeCenter JS22 is comprised of two IBM POWER6 multi-core processors. In comparison to the UltraSparc T2 the IBM POWER6 microarchitecture [6] follows a contrary architectural approach. Ultra high frequency core design (up to 4.7Ghz) optimised for ILP is favoured over massive chip multithreading (that is TLP). Based on the POWER5 microprocessor, the POWER6 architecture implements two high-frequency *simultaneous multithreading* (*SMT*) cores per chip. SMT enables simultaneous execution of up to two threads per core. With its 4MB core-private L2 caches and a large shared 32MB L3 cache the POWER6 microarchitecture is optimised for performance and computational power. The SMP interconnect facilities enable system configurations of up to 32 POWER6 processors.

Operating system and environment. Our goal was to evaluate the systems under realistic conditions. To achieve this, we used typical vendor setups where applicable. This means, that we used the vendor's operating system and compiler tool-chain (see Table 1), and allowed maximum compiler optimisation. For OpenMP and MPI compilation the corresponding tools and flags in the compiler tool-chain were used.

Scheduling policies. Consistent with our goal to use realistic conditions, we did not manipulate the operating system scheduling policies. With the OpenMP scheduler we also didn't change the policy, except for comparison runs with the NAS Benchmarks on the T2+, where we were interested in a single chip performance evaluation. Since the systems were delivered with SMP dual- chip configurations we needed to delegate the execution of OpenMP [11] threads to a specific processor. This was achieved via the native task schedulers.

3 Benchmarks

In this section we describe the software used for performance analyses. We use a collection of *microbenchmarks* for performance evaluation of individual system components (such as the memory subsystem). The *NAS Parallel Benchmarks* serve to evaluate the performance of typical building blocks that frequently appear in computational science applications. As an example of scientific applications, we use an ocean simulation code with a hybrid MPI/OpenMP programming model.

Microbenchmarks. The *STREAM Memory Benchmark* [8] is a synthetic benchmark that measures the performance of four operations on large vectors: (1) *copy*, (2) *add*, (3) *scale* (involves multiplication operations), and (4) *triad* (combines *copy*, *add* and *scale*). The main goal of the STREAM benchmark is a realistic evaluation of sustainable memory performance. *Cache Bench* is a specialised memory benchmark designed for evaluation of memory hierarchy performance. Different tests are used including *read*, *write*, *modify* operations and the *memset()* and *memcpy()* functions from the C library. Each test is executed using several vector sizes [9]. *DGEMM* [3] tests the floating point performance of a double precision Matrix-Matrix multiplication via hardware optimised Level-3 Blas routines [2].

NAS Parallel Benchmarks. This benchmark suite was originally developed by the Numerical Aerodynamic Simulation (NAS) program at the NASA Ames Research Center for effective evaluation of platforms for computational fluid dynamics (CFD) applications. The NAS parallel benchmark (NPB) suite [1] has proven to be a credible solution for measuring computational performance of parallel computer systems. In our performance evaluation study we use a C version of NPB 2.3 parallelised with OpenMP [11] that was developed by the Omni OpenMP Compiler Project [10]. We have selected four NPB benchmarks: FT, CG, LU, and EP. The FT benchmark executes a 3-D fast Fourier Transform (FFT) based kernel. In the CG benchmark a Conjugate Gradient method is used to approximate the smallest eigenvalue of a sparse matrix. The EP (Embarrassingly Parallel) kernel generates pseudo-random numbers and can be used as an estimation of the systems scalability. In the LU application benchmark a block lower triangular-block upper triangular system of equations is solved [1]. In our performance experiments all benchmarks have been executed with the problem size indicated as *Class B*.

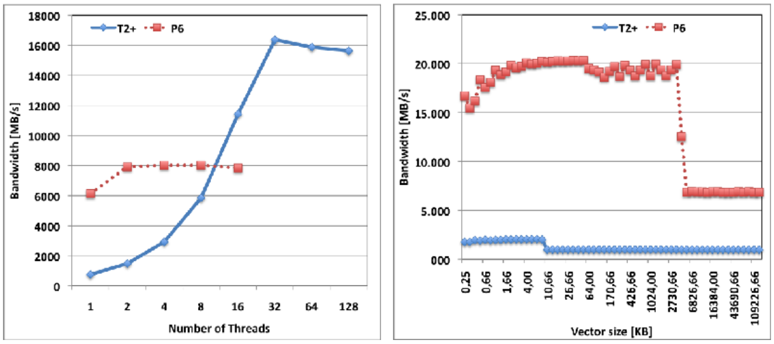
Ocean Simulation. This is a real-world hybrid MPI/OpenMP Fortran90 code [4] that simulates the western intensification of wind-driven ocean currents [16]. A 2D Stommel Model of Ocean Circulation (2D-SMOC) is solved by using a five-point stencil and Jacobi iteration. We use this application to complement the performance evaluation results of Micro- and NAS benchmarks.

4 Results

In this section we present the performance evaluation results obtained using micro benchmarks, NPBs, and the ocean simulation application. Our primary aim is to evaluate the suitability of the T2+ processor for computational science. We use the performance results of the POWER6 processor for comparative purposes only.

4.1 Microbenchmarks

Memory access is a limiting factor of the performance of many scientific computing applications. For an estimation of sustainable memory performance we used the STREAM memory benchmark (see Section 3) with a variable number of threads. All operations were executed on large vectors. The performance results obtained are depicted in Figure 2(a). With increased thread count the T2+ based system is able to constantly increase overall memory bandwidth until up to 16 threads. On the POWER6 system the memory performance gain in multithread runs is less intense. We may observe that for single thread runs the memory bandwidth of the T2+ based system is about five times lower compared to the POWER6.

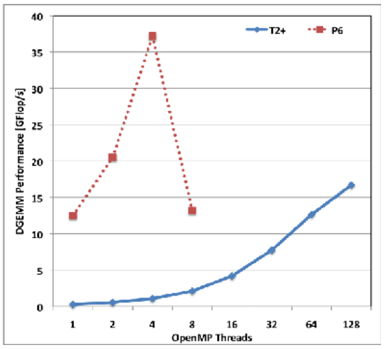


(a) STREAM (Triad) (b) Cachebench

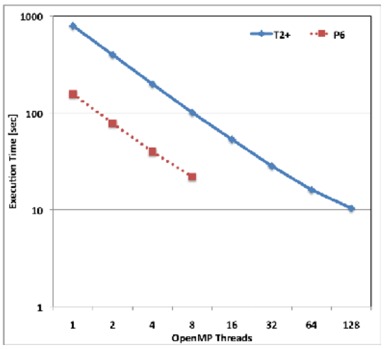
Fig. 2. Performance evaluation of the memory subsystem using STREAM and Cachebench

Figure 2(b) depicts the evaluation of memory bandwidth using Cachebench for various vector sizes. Data was acquired in single thread runs of the read/modify/write test with vector lengths between 256 byte and 100MB. A bandwidth drop can be seen between the L1 (8KB) and the L2 cache on the T2+. The change between the L2 cache and the main memory is less distinctly observable via this test for the T2+. On the POWER6 based system we see drops between L1 and L2 cache at 64KB and between L2 and the main memory at the L2 cache size of 4MB. We may observe that Cachebench performance results of T2+ are significantly lower than POWER6 for all tested vector sizes.

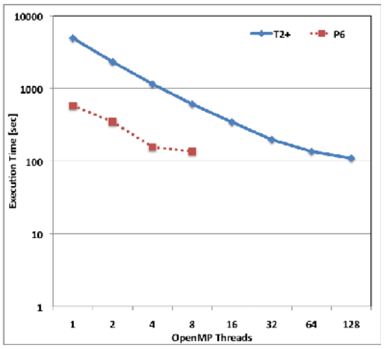
Figure 3(a) depicts the evaluation of sustainable floating point performance using *DGEMM* benchmark. This benchmark uses system vendor SMP optimised Level3-Blas subroutines with a fixed matrix dimension and a variable number of threads. The benchmark results confirm the design principles of the T2+ system with low single thread performance but excellent scaling capabilities. The maximum performance of 37.18 GFlop/s is achieved on the POWER6 system in a 4-thread run. For the *DGEMM* benchmark the use of SMT on POWER6 system shows a steep performance drop when the number of SMT threads is larger than the number of physical cores. This is contrary to the T2+ system where a maximum performance of 16.69 GFlop/s is observable at the maximum number of 128 available SMT threads (please note that our evaluated T2+ based system has 16 physical cores).



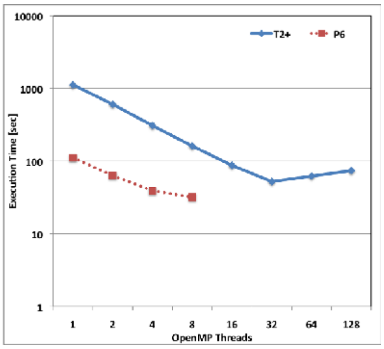
(a) DGEMM Performance in GFlops



(b) NPB EP Execution Time



(c) NPB LU Execution Time



(d) NPB FT Execution Time

Fig. 3. Performance results for the DGEMM, NPB EP, NPB LU, and NPB FT benchmarks

4.2 NAS Parallel Benchmarks

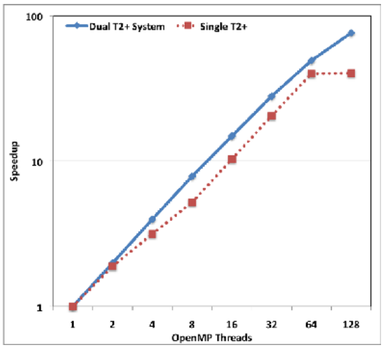
In this section we investigate the performance of a subset of the NPB benchmarks on the T2+ and POWER6 based systems. The aim is to identify NPBs that perform well on T2+ based systems. We used the problem size that in NPB is indicated as *Class B* in all presented experiments. On the T2+ based system the maximum number of SMT threads was set equal to the number of available logical CPUs (128). On the POWER6 based systems we performed experiments up to the limit of 8 SMT threads. With these settings we could investigate performance of the systems under demand for heavy resource sharing and per core context switching.

Figure 3(b) depicts performance evaluation results obtained using the NPB EP benchmark. The NPB EP benchmark generates pairs of Gaussian random deviates. This *embarrassingly* parallel task is obviously very well suited for the massively multithreaded T2+. The T2+ based system exhibits a nearly linear speedup. We may observe that the highest achievable performance of the NPB EP benchmark on T2+ based system is higher than on POWER6. This is due to the larger number of cores and better SMT capabilities of T2+. But, the POWER6 system clearly outperforms the T2+ in single thread performance and computational core power.

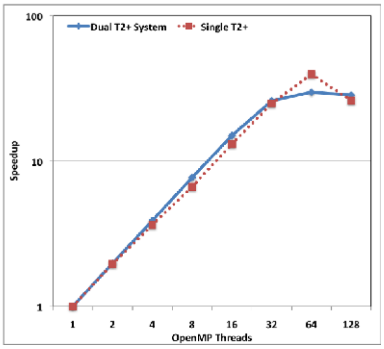
The performance evaluation results obtained using the NPB LU benchmark are depicted in Figure 3(c). NPB LU benchmark uses a symmetric successive over-relaxation (SSOR) method to solve a seven-block-diagonal system. A pipelined algorithm is used to solve the triangular systems [5]. We may observe in Figure 3(c) that for large numbers of threads the T2+ based system outperforms the POWER6. However, due to the LU synchronisation overhead, the increase of performance with the number of threads is not as high as for EP.

Figure 3(d) depicts performance evaluation results obtained using the NPB FT benchmark. The FT benchmark performs a 3-D FFT by execution of three 1-D FFT routines. Therefore the 3-D data is copied into 1-D work arrays for each dimension. Before the final 1-D FFTs are performed, array transposes take place [13]. With elevated communication demand the scaling abilities of the T2+ are limited. For FT benchmark the T2+ is unable to reach the POWER6 system's performance in any of our test cases. We may observe that on the T2+ system almost no speedup can be achieved with more than 32 threads.

Figure 4 depicts the speedup of the EP and CG benchmarks on the T2+ based system. For this evaluation we compared the speedup of the system when both T2+ chips are used with the case when only one of the available two chips is used. Although we ran these tests with all problem sizes, only *Class B* tests will be discussed here. The *Class B* provided the most interesting results, by being able to utilise the available resources.



(a) Speedup NPB EP



(b) Speedup NPB CG

Fig. 4. The speedup of EP and CG benchmarks on the T2+ based system. Speedup when both available T2+ processors are used is compared to speedup when a single T2+ processor is used.

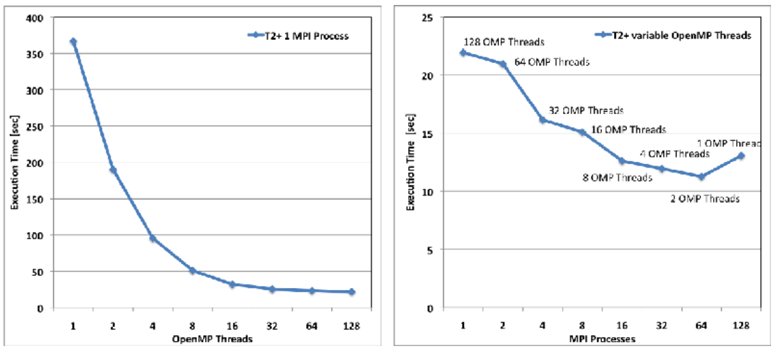
The speedup of EP benchmark is depicted in Figure 4(a). If both T2+ chips are used, the EP benchmark shows the best speedup of 76.15 when executed with 128 threads. Up to 32 threads the speedup is nearly linear. If only one of the available two T2+ chips is used, the speedup is slightly lower and reaches its maximum with 64 threads for both benchmarks. For more than 64 threads there is no performance improvement if a single T2+ chip is used, but also no dramatic performance drop. This is due to the embarrassingly parallel nature of the EP benchmark.

Figure 4(b) depicts the speedup of the CG benchmark. The CG benchmark uses a Conjugate Gradient method to approximate the smallest eigenvalue of a sparse matrix.

Due to the communication demand of the CG benchmark the single chip run can take advantage of data locality. For 64 threads the execution of the CG benchmark on a single chip shows better speedup and performance (measured in absolute execution time) than the case when both chips are used. For more than 64 threads there is a clear performance drop for both cases. In this experiment the SMP interconnect seems to be a bottleneck when a high number of threads is used.

4.3 Ocean Simulation

In this section we investigate the performance of a real-world Ocean Simulation application on our T2+ based system. This application is implemented in Fortran90 and parallelised using MPI and OpenMP (see Section 3). While the initial data partitioning is done via message passing, the Jacobi iterations are parallelised with OpenMP [11]. The nature of this application allows an investigation on the impact of combination of these programming models. Therefore, we conducted multiple performance experiments for various numbers of MPI processes and OpenMP-threads. Figure 5 depicts the performance results obtained on the T2+ based system.



(a) OpenMP (b) OpenMP/MPI

Fig. 5. Performance results for the ocean simulation application

In Figure 5(a) are presented performance results obtained when the number of MPI processes is constant (a single MPI process is specified) and the number of OpenMP threads is varied from one to 128. The results reveal that this application scales well on the T2+ based system when the number of OpenMP threads is varied. The best result (21.92 seconds) is achieved when all 128 OpenMP threads are used.

Figure 5(b) depicts the performance results that are obtained for various combinations of the number of MPI processes and OpenMP threads, where the sum of MPI processes and OpenMP threads is kept constant at 128. The best result (11.25 seconds) is obtained for 64 MPI processes each having two OpenMP threads.

5 Related Work

There exist many different techniques for performance evaluation of highly parallel computer systems. Measurement-based performance evaluation approaches are commonly used for the evaluation of existing computer architectures, while the model-based approaches are typically used to answer what-if questions for future architectures that are under the development. In [12], a measurement-based approach is used for the evaluation of the Intel Woodcrest processor for scientific computing using a set of benchmarks. An evaluation of the STI Cell/B.E. processor using scientific computing kernels is presented in [20]. In [7], a model-based approach is used for performance evaluation of bioinformatics applications on GPU-accelerated architectures. In [19], a sparse matrix-vector multiply kernel is optimised for different multicore platforms. This approach may have the benefit of discovery of algorithmic optimisations for a specific hardware.

In this paper we present a comprehensive measurement-based performance analysis study of the T2+ processor, which aims at investigating the suitability of this processor for computational science. For this purpose we use, alongside the NAS parallel benchmarks, a collection of microbenchmarks and a real world ocean simulation application.

6 Conclusions

CMT systems like the SUN UltraSparc T2+ are especially well suited for server workloads with a high level of thread parallelism. In our experimental evaluation for the scientific computing domain we observed excellent scaling capabilities on workloads with little or no inter-thread communication such as the NPB EP benchmark. However, for problems with higher communication demand (such as NPB CG benchmark) or unpredictable memory access patterns the T2+ shows performance drawbacks. Due to the high number of threads, increased communication overhead becomes more severe on the T2+ than on high frequency multi-core designs like the IBM POWER6. This issue is further aggravated in SMP configurations where multiple T2+ processors are used. We have observed that for communication-intensive codes such as NPB CG, better performance may be achieved using only one of the two available T2+ processors. Using a real-world ocean simulation hybrid code we conducted multiple performance experiments for various numbers of MPI processes and OpenMP-threads. The best result was obtained for a combination of a rather larger number of MPI processes with a small number of OpenMP threads.

In the future we plan to investigate the performance of emerging multi-core systems using model-based evaluation techniques.

Acknowledgements. The authors are grateful to Martin Wimmer for numerous discussions and helpful comments regarding the work presented in this paper.

References

1. Bailey, D., Harris, T., Saphir, W., Van der Wijngaart, R., Woo, A., Yarrow, M.: The NAS Parallel Benchmarks 2.0. NAS Technical Report NAS-95-020 (December 1995)
2. Dongarra, J., Du Croz, J., Duff, I.S., Hammarling, S.: A set of Level 3 Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software (March 1990)

3. DGEMM Benchmark, `https://computecanada.org/?pageId=138`
4. Kaiser, T.H., `http://www.sdsc.edu/~tkaiser/stommel.html`
5. Jin, H., Frumkin, M., Yan, J.: The OpenMP Implementation of NAS Parallel Benchmarks and its Performance (1999)
6. Le, H.Q., Starke, W.J., Fields, J.S., O'Connel, F.P., Nguyen, D.Q., Ronchetti, B.J., Sauer, W.M., Schwarz, E.M., Vaden, M.T.: IBM POWER6 microarchitecture. IBM J. Res. and Dev. 51(6) (November 2007)
7. Liu, W., Schmidt, B., Mueller-Wittig, W.: Performance Analysis of General-Purpose Computation on Commodity Graphics Hardware: A Case Study Using Bioinformatics. Journal of VLSI Signal Processing Systems 48(3), 209–221 (2007)
8. McCalpin, J.D.: Memory Bandwidth and Machine Balance in Current High Performance Computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter (December 1995)
9. Mucci, B., London, K., Thurman, J.: The CacheBench Report (November 1998)
10. NAS Parallel Benchmarks in OpenMP, `http://phase.hpcc.jp/Omni/benchmarks/NPB/index.html`
11. OpenMP Application Program Interface. Version 3.0 (May 2008), `http://www.openmp.org/mp-documents/spec30.pdf`
12. Roth, P.C., Vetter, J.S.: Intel Woodcrest: An Evaluation for Scientific Computing. In: Proc. 8th LCI International Conference High-Performance Clustered Computing (2007)
13. Saphir, W., Van der Wijngaart, R., Woo, A., Yarrow, M.: New Implementations and Results for the NAS Parallel Benchmarks 2. In: 8th SIAM Conference on Parallel Processing for Scientific Computing (1997)
14. Shah, M., Barreh, J., Brooks, J., Golla, R., Grohoski, G., Gura, N., Hetherington, R., Jordan, P., Luttrell, M., Olson, C., Saha, B., Sheahan, D., Spracklen, L., Wynn, A.: UltraSPARC T2: A Highly-Threaded, Power-Efficient, SPARC SOC. Sun Microsystems (2007)
15. Spracklen, L., Abraham, S.G.: Chip Multithreading: Opportunities and Challenges. In: 11th International Symposium on High-Performance Computer Architecture (2005)
16. Stommel, H.: The western intensification of wind-driven ocean currents. Transactions American Geophysical Union. 29, 202–206 (1948)
17. Sun SPARC Enterprise T5140 Server, `http://www.sun.com/servers/coolthreads/t5140/`
18. SUN SPARC Enterprise T5120, T5220, T5140 and T5240 Server Architecture. White Paper. Sun Microsystems (April 2008)
19. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., Demmel, J.: Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing (2007)
20. Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, A.K.: Scientific Computing Kernels on the Cell Processor. International Journal of Parallel Programming 35(3), 263–298 (2007)

Streamlining Offload Computing to High Performance Architectures

Mark Purcell[1], Owen Callanan[1], and David Gregg[2]

[1] Dublin Software Lab, IBM Ireland
[2] Trinity College Dublin, Ireland
{mark_purcell,owen.callanan}@ie.ibm.com,david.gregg@cs.tcd.ie

Abstract. Many important scientific, engineering and financial applications can benefit from offloading computation to emerging parallel systems, such as the Cell Broadband Engine™(Cell/B.E.). However, traditional remote procedure call (RPC) mechanisms require significant investment of time and effort to rewrite applications to use a specific RPC system. As a result, offloading functions to remote systems is not viable for many applications. IBM® Dynamic Application Virtualization ™(DAV) insulates the application developer by automatically generating stub libraries that allow direct calling of remote procedures without application source code modification. In this paper, we describe DAV automates the conversion of client applications to use remote procedure calls. DAV can generate stub libraries for a wide variety of client applications running on a variety of architectures, allowing allows simple and fast remote procedure call enablement of applications with minimum programming effort.

Keywords: Offload computing, computer architecture, parallel computing, remote procedure call.

1 Introduction

Offloading computation onto remote computer systems is a useful way to accelerate computationally intensive applications. For example, in the financial services sector, spreadsheet applications are used to evaluate options prices using the Black-Scholes formula. This is a computationally intensive algorithm which can be significantly accelerated on specialised processors such as the Cell/B.E. [2]. Offloading the Black-Scholes formula onto remote high performance systems significantly improves performance of the spreadsheet, allowing faster response to financial market conditions. Offloading calculations from the application to the remote library is a significant challenge in the industrial setting however.

Traditional remote procedure call (RPC) mechanisms enable applications to call functions from libraries running on remote machines. These systems require the client application to use a specific API, forcing an application rewrite. The time and effort required is significant and, as a result, offloading functions using traditional remote procedure call mechanisms is often not viable. Such RPC mechanisms can also create maintenance problems since changes in the API require rewriting the client applications. In the industrial domain, these problems are a major barrier to offloading computation to systems optimised for particular types of processing, such as machines based on the Cell/B.E. processor, which is heavily optimised for numerical computing [2].

The Virtualizer component of IBM Dynamic Application Virtualization (DAV) addresses this problem. Based on library and function specific tags supplied by the library developer, the Virtualizer generates libraries that exactly mimic the interface of the local machine libraries. As a result, no application code changes are required to offload functions to remote machines using DAV. The client application need only relink to the Virtualizer generated libraries, instead of the native code libraries. The Virtualizer currently generates stub libraries C, C++ and Java stub libraries. These stub libraries are generated from simple tags in a small domain-specific language. Furthermore DAV can be extended to support other languages and applications if required.

This paper presents an overview of DAV and the Virtualizer tool and we discuss the efficacy of using generative code techniques to eliminate application code changes when offloading functions to remote machines. The paper starts with an overview of the DAVsystem in section 2.1. Section 3 then continues with a discussion of the problems faced in automatically generating libraries that handle variables and function parameters with unknown size and usage. In section 4 we describe how Virtualizer uses a simple domain-specific language to describe the size and usage of parameters, and a generator to automatically generate interface code for remote procedure calls. Finally in section 5, a brief discussion of a traditional remote procedure call system, RPCGen, is included.

2 DAV Overview

2.1 DAV Architecture

The default DAV data transport system provides a flexible, extendable, light weight, low-latency infrastructure to allow client applications to access libraries on remote systems, giving clients access to multiple remote libraries, or services, through a central broker. Services register availability with the broker, which then maintains information on the available services. To increase availability of a service, multiple instances of a service can be run concurrently on multiple machines. In order to access an DAV service, client applications send a service request to the broker and the broker then chooses the best available service node to process the request. The client processes its service request by interacting directly with this service node. DAV services can run on various high performance architectures including x86 processors and the Cell/B.E. processor. Thus the broker provides a single interface to multiple services, running on multiple nodes with heterogeneous architectures.

DAV can use other transport systems. For example, DAV could use a third party grid scheduling system. This is possible since DAV defines a simple interface API.

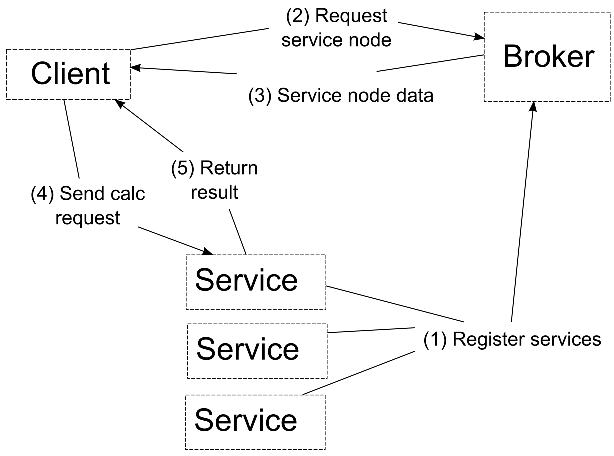


Fig. 1. DAV Architecture

Any transport subsystem that implements this API can be used by DAV. By implementing a wrapper interface for this grid scheduling system, an organisation can use DAV to allow their applications to easily offload calculation to remote machines, whilst utilising their investment in their existing grid infrastructure.

The DAV code generator generates client-side stub libraries that duplicate the interface of the original native code libraries. These libraries enable client applications to call remote library functions without changing the application source code. For an application to use DAV services, the library developer must write tags, which are included in the library header file, for all the functions in the library to be offloaded. The Virtualizer takes the modified header file as input and uses the DAV tags to generate client-side stub libraries and server-side skeleton libraries. The server-side skeleton libraries are automatically deployed to the remote server using the DAV deployment tool. To access the offloaded library, the client application need only be linked to the generated client stub libraries instead of to the original native code libraries. Once the DAV service has been started and registered with the broker, the offloaded library is available for use.

Traditional function offload systems require that the client application is rewritten to use a specific API; a process which is often difficult and time-consuming. The resulting code can be difficult to maintain since changes to the API require rewriting of the client application. Automatically generating libraries that handle all communication to and from the DAV services addresses these problems, making DAV-enabled applications easy to use and maintain. If the DAV infrastructure changes then the developer need only regenerate the stub libraries and relink the application.

2.2 DAV Application Areas

Performance is is often limited by the hardware platform running an application. For example, a spreadsheet performing computationally intensive data analysis is limited by the desktop computer it is run on. Meanwhile, specialised high performance computing systems such as the Cell/B.E. processor can deliver significant performance improvements over standard processors for many common computing kernels [3,8]. Larger multiple processor machines such as clusters of x86 or Cell/B.E. processors now provide significant processing power and DAV offers an excellent solution to accessing the performance of these systems from existing applications.

DAV is initially targeted at the financial services sector, but it has applications in other fields of business, science and engineering. For example many science and engineering applications are based on standard linear algebra packages such as BLAS [4]. The Cell/B.E. delivers significant performance for many of the core BLAS routines, and DAV provides a straightforward way to access this performance.

3 RPC Challenges

Many languages, including C, do not have self-describing data structures. Given a pointer to a data structure, the language doesn't know the size or shape of the data structure. Pass-by-reference array parameters are an example of this problem. Unless the size of the array is known at compile time, then extra information must be passed to the function to indicate the size of the array. An example of this is shown in figure 2. Unknown parameter sizes are a problem for remote procedure offload systems such as DAV, since the operand data must be transferred to the remote server and unless the size of the data is known, data transfer is not possible. Thus some mechanism is required to specify the size of pass-by-reference parameters to allow DAV to generate correct stub libraries.

Pass-by-reference parameters can also be modified within a function, or a function may allocate memory to a pointer passed as a parameter. As a result, operand data may only need to be sent, retrieved, or sent and retrieved. The C language syntax does not specify how a pointer passed to a function is used inside the function and, to maximise efficiency, data should only be transferred when necessary. Again, a mechanism is required to specify the transfer direction of pass-by-reference parameters.

4 A Solution: DAV Tags

DAV tags provide a solution to automatic transfer of pass-by-reference parameters. Using these tags, the library developer can specify that a function is to be hosted in the remote library, and the size and transfer direction of any pass-by-reference parameters. DAV tags are based on the DOxygen/JavaDoc tags used to specify documentation information within application source code. DAV tags can also pass other information to the Virtualizer, such as the library name or information about structs. The format of the DAV tags is as follows:

```
/**IBMDAV*
@tagType value
@property value
@property value
... */
```

There are three main tag types:

1. Library tags specify settings for the entire library including adding prefixes or suffixes to DAV exported functions.
2. Function tags set properties for specific functions, including the size and transfer direction of pass-by-reference parameters and return values.
3. Struct tags are used to inform the Virtualizer about any structs used by DAV exported functions, including the size of any pointer type struct members.

Various property tags are used by the three tag types, and are shown in table 1.

Table 1. DAV semantic property tags

Property Tag	Purpose	Used with
@library <name>	Library name	-
@func <name>	Virtualize function <name>	-
@struct <name>	Describe struct <name>	-
@param[in|out|inout]<name>	Function-param details	@func, @struct
@return	Specify size of returned data	@func
@dimensions [<size>]	Specify size of an array	@param, @return
@ type string	Denote string parameter	@param, @return
@prefix <p>, @suffix <s>	Function prefix or suffix	@library
@lib_options "<options>"	Additional linker options	@library

Figure 2 shows a C function that takes two pointers to arrays as operands, along with an integer to specify the size of the arrays. Also shown in figure 2 are the tags required for the Virtualizer to create stub libraries for the function. The Virtualizer handles the integer parameter *s*, automatically because it is a scalar type, but extra information is required for the two pointer parameters to describe the data that they point to. In this case, they are both pointers to arrays of size *s*. The first array, *a*, is input only, whilst the second, *z*, is both an input and an output since it is modified by the function. The *@param* and *@dimensions* tags are used to pass this information to the Virtualizer.

Figure 3 shows the code generation process used by the Virtualizer. The user runs the Virtualizer from the command line, passing the library header file including tags, as input. The Virtualizer checks the syntax of the C header definitions and of the tags, and parses the header file using the Eclipse C/C++ parser, CDT. Function prototype information is extracted and stored in a temporary internal representation. The tags are extracted from the header file using

```
/**IBMDAV* @function calcArray
  @param[in] a @dimensions[s]
  @param[inout] z @dimensions[s] */
double calcArray( double *a, double *z, int s ){
  double res = 0;
  for (int i = 0; i < size; i++) {
    z[i] = a[i]{*}2;
    res += z[i];
  }
  return res;
}
```

Fig. 2. Function with DAV tags for unknown parameter size and transfer direction

CDT and passed to the Java parser-generator, Javacc, which extracts the information from the tags and stores it. The Virtualizer processes the stored tag and the header file information, combining them into a single XML document. Using XSLTs, the Virtualizer produces the source code for both client-side stub libraries and server-side skeleton libraries. Finally the client-side stub library is compiled on the client system. The server-side skeleton library is then deployed to the machine that will run the service using the DAV deployer tool.

When run using the tag information shown in figure 2, the Virtualizer will produce a client-side stub library that exports a *calcArray* function with identical syntax to the original native code function. This function consists of code to construct the transportable data, manage calling of the remote DAV function and extract the returned result data. All interactions with the underlying infrastructure are completely contained by the generated function, so no code changes to the client application source are required.

By using generative code techniques DAV supports a variety of client types including C, C++, Java and VBA applications. C/C++ are supported for server side libraries. All standard basic types in each language are supported as well as strings, arrays, two dimensional arrays and data structures. Pointers to all of these types, including pointers to arrays of up to two dimensions, are also supported. All supported types are natively supported, no DAV specific types are required, so DAV requires no client side code changes of any kind.

5 Related Work

RPCGen is an example of a traditional API based RPC mechanism [5]. RPCGen simplifies the use of RPC by generating stub libraries that wrap much of the RPC API. However RPCGen still requires significant code changes to the client application. RPCGen remote procedures use a different function call syntax to the original native code functions, and some parameter types, such as variable length arrays, must use RPC specific data types instead of the native C types. As a result the client application code must be changed to allow RPCGen remote procedures to be used. A significant amount of work is required to convert

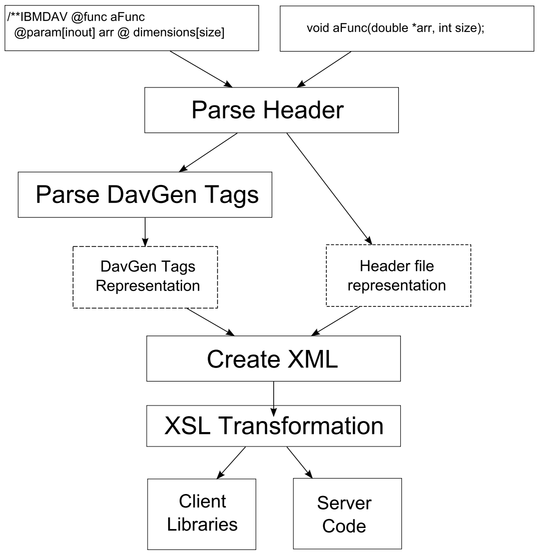

/**IBMDAV @func aFunc
@param[inout] arr @ dimensions[size]
void aFunc(double *arr, int size);
Parse Header
Parse DavGen Tags
DavGen Tags Representation
Header file representation
Create XML
XSL Transformation
Client Libraries
Server Code

Fig. 3. Code generation process

```
double *matrix = new double[N*N];
for ( int i = 0; i < N; i++ ) {
  matrix[i] = i;
}
result res;
result *resPtr;
double sum = summat(&m, *N, &resPtr);
```

Fig. 4. Original Native Source Code

an application to use RPCGen remote procedures, as shown by comparing the original source in figure 4 with the source modified to use RPCGen in figure 5.

CORBA is an object request broker system that allow applications written in different languages to exchange data objects [1]. CORBA uses a language-independent interface definition language (IDL) to define the interface between client applications and the object broker system. This interface code is then compiled into language-specific stub and skeleton libraries using the IDL compiler provided with a CORBA implementation. The use of an interface definition language allows CORBA implementations to insulate the application developer from some of the complexity of the CORBA object broker system. However substantial code changes are still required, as can be seen in figure 6. Component Object Model (COM) from Microsoft is another technology that allows different applications to interact though a common object format [6]. In a similar fashion to CORBA, COM uses an IDL to define the interface between applications and the COM data transport system.

```
double *matrix = new double[N*N];
for ( int i = 0; i < N; i++ ) {
  matrix[i] = i;
}
matrices m;
m.matrix_len = N*N;
m.matrix_val = matrix;

result res;
result *resPtr;
double sum = *summat(&m, *N, &resPtr);
```

Fig. 5. Example of use of RPCGen specific types for 2d matrix function

```
CORBA_ORB_var = CORBA_ORB_init();
ifstream in(Example.ref);
char s[1000];
in >> s;
CORBA_Object_var obj = orb->string_to_object(s);
Example_var p = Example::_narrow(obj);

double *matrix = new double[N*N];
for (int i = 0; i < N; i++) {
  matrix[i] = i;
}
result res;
double sum = p->summat(matrix, N, &res);
```

Fig. 6. Example of CORBA code changes for a 2d matrix function

Automatically partitioning an application to distribute it across multiple systems is a related problem to the one we address. An example is J-Orchestra [7] which can automatically partition and distribute Java bytecode programs to execute just as if they were running on a single system. J-Orchestra works by rewriting the Java bytecode to insert an extra level of indirection into all object references. By intercepting memory accesses in this way, references to objects on remote systems can be redirected, and similarly local method invocations can be replaced by remote procedure calls. It is important to note that J-Orchestra only works because Java has well-behaved references and self-describing data structures. Redirecting all memory references would be much more difficult in C/C++ because general pointers can be used to access arbitrary parts of memory.

6 Experimental Results

To maximise the performance gain from calculation offload to a remote system using DAV, the offload overhead must be minimised. There are two main causes

Table 2. DAV latency performance data

Performance Test	Latency (μs)
Local Transport	59.9
DAV Transport (localhost interface)	260
DAV Transport (remote server, Gigabit Ethernet)	406

of overhead in an DAV request. The first is the data marshalling overhead which is the time required to pack and unpack request and result data into a transportable format. The second overhead is the cost of transporting the data to the remote system. To measure the data marshalling overhead, a different transport subsystem is used, which is called the local transport. The local transport is a skeleton transport that processes requests on the client machine, and does not use the network stack at all. Measuring the time required to process requests using the local transport allows measurement of the calculation overhead from using DAV to process requests. The network stack overhead is measured using the standard DAV transport by running the client and server on the same machine, accessing the server through the network stack localhost interface. Finally the latency is measured for a client accessing a remote service over a typical network. Latency figures are presented for these three scenarios. The test function used is shown in figure 2. The operand arrays contain 4 elements each. As a result the time required to execute the test function is very small compared to the overall time required to process a remote request. The client machine is a 2.4 GHz Intel Core 2 processor running Windows, and the server machine is a 3.0 GHz Intel Xeon processor running Linux.

The performance figures in table 2 show that processing required for a single small DAV request is low, and overall takes less than 60 microseconds. Processing an DAV request using a service running on the client machine, accessed through the localhost network interface adds a further 200 microseconds to this figure. This reflects the increased processing overhead caused by moving data through the network stack. Finally, processing the DAV request over a Gigbit Ethernet network takes a little over 400 microseconds, which shows the cost of transporting the request and result data across the network.

Functions with high computational complexity are easier to accelerate, since the offload overhead is small compared to calculation time. For large data volumes, the speed of the network system is the critical factor in determining offload overhead, since the cost of making the remote function call is small compared to the cost of transporting the data. For small data volumes, where the data transport cost is low, the cost of the function call is comparitvely large. We can see from the above figures that DAV has a low function call overhead, allowing DAV to accelerate even relatively small calculations.

7 Conclusion

Traditional RPC mechanisms require significant application code changes to offload functions onto remote machines. This a major barrier to adoption of offload computing in industry, because customers are reluctant to significantly modify their software. DAV's Virtualizer tool addresses this issue and eliminates any application code changes when converting to remote procedure calls, substantially simplifying the process of converting an application. The Virtualizer uses a straightforward set of tags to allow the developer to supply information about unknown variables and function parameters such as pointers to arrays. When supplied with these tags in the library header file, the Virtualizer generates source code for client stub libraries and server skeleton libraries that wrap all interactions with the DAV infrastructure, completely insulating the client application. The external interface of the client-side stub library is identical to the original native code library, so no application code changes are required to enable an application to use remote procedures. Furthermore the client application is protected from any changes to the middleware API. If the API changes, then the source code for the libraries need only be regenerated, and the application rebuilt using the new libraries. This greatly reduces the effort required by industrial users of DAV to offload computations from the desktop to more powerful cluster computers. The use of domain-specific generative code techniques allows DAV to substantially reduce the cost and effort required to write and maintain applications that offload calculation to remote systems.

References

1. Common object request broker architecture: Core specification. Technical report, Object Management Group (2004)
2. Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R., Shippy, D.: Introduction to the cell multiprocessor. IBM J. Res. Dev. 49(4/5), 589–604 (2005)
3. Kurzak, J., Buttari, A., Dongarra, J.: Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization–LAPACK Working Note 184. LAPACK Working Note 184 (May 10, 2007)
4. Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T.: Basic linear algebra subprograms for fortran usage. ACM Trans. Math. Softw. 5(3), 308–323 (1979)
5. Rago, S.A.: UNIX System V network programming. Addison-Wesley Longman Publishing Co., Inc., Boston (1993)
6. Rogerson, D.: Inside COM. Microsoft Press, Redmond (1997)
7. Tilevich, E., Smaragdakis, Y.: J-orchestra: Automatic java application partitioning, pp. 178–204. Springer, Heidelberg (2002)
8. Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, K.: Scientific computing kernels on the cell processor. International Journal of Parallel Programming 35(3), 263–298 (2007)

Multi-walk Parallel Pattern Search Approach on a GPU Computing Platform

Weihang Zhu and James Curry[*]

P.O.Box 10032, Department of Industrial Engineeirng,
Lamar University, Beaumont, Texas, 77710, USA
Weihang.Zhu@lamar.edu, jcurry@my.lamar.edu

Abstract. This paper studies the efficiency of using Pattern Search (PS) on bound constrained optimization functions on a Graphics Processing Unit (GPU) computing platform. Pattern Search is a direct search optimization technique that does not require derivative information on non-linear programming problems. Pattern Search is ideally suited to a GPU computing environment due to its low memory requirement and no communication between threads in a multi-walk setting. To adapt to a GPU environment, traditional Pattern Search is modified by terminating based on iterations instead of tolerance. This research designed and implemented a multi-walk Pattern Search algorithm on a GPU computing platform. Computational results are promising with a computing speedup of 100+ compared to a corresponding implementation on a single CPU.

Keywords: Nonlinear Optimization, Pattern Search, GPU, CUDA.

1 Introduction

Graphics Processing Unit (GPU) computing is an emerging technology of parallel computing due to its low cost per instruction. Modern GPU computing technology features a ‘Single Instruction – Multiple Threads’ mode, which is amenable to heavy and repetitive computation. Researchers have employed GPU in many fields such as physically-based simulation, financial engineering, and image/video processing [8]. Several researchers have employed GPU for optimization. Li et al. (2006) studied parallel Particle Swarm Optimization [6]. Zhu et al. (2008) examined parallel Tabu Search for the Quadratic Assignment Problem [11]. This paper examines using the GPU for unconstrained nonlinear optimization with bound constraints with Pattern Search, a direct search method commonly found in non-linear optimization. The objective is to minimize a nonlinear function *f(x)* subject to range constraints that $a_i \le x_i \le b_i$ where $x \in \Re^n$ and $f : \Re^n \to \Re$. Such global optimization problems over continuous spaces are ubiquitous throughout the scientific community. When the objective function is non-linear and non-differentiable, direct search approaches are the methods of choice.

[*] Corresponding Author.

Direct search methods are characterized by neither requiring nor explicitly approximating derivative information. Direct search methods are considered as 'zero-order methods', which are different from 'first-order methods' such as steepest descent method, or 'second-order methods' such as Newton's method. The 'order' of a method indicates the highest order term being used in the local Taylor series approximation to the nonlinear function *f*. With 'zero-order', direct search methods work directly with *f*, instead of with a local approximation to *f*.

Pattern Search methods are a subset of direct search methods. Pattern Search is traditionally employed when the gradient of the function is not reliable when performing the search [3], [4], [9]. This robustness makes Pattern Search a safe choice for solving many different types of problems [1]. Parallel Pattern Search methods have been proposed in the past research in CPU platforms [2], [3]. However, instead of running multiple simultaneous searches, their methods assign the individual search direction to CPUs within the pattern search. In this paper, we present a 'Single Instruction – Multiple Threads – Pattern Search' (SIMT-PS) algorithm developed on a GPU platform. This algorithm takes the approach of starting multiple simultaneous Pattern Searches from massive random starting points to generate a large computation tasks to the GPU. Significant speedup in computation and optimization speed can be achieved with this promising approach, as suggested by our computational experiments results.

The remainder of this paper is organized as follows. Section 2 presents background information on the GPU computing. Section 3 provides an overview of the Pattern Search. Section 4 discusses the analysis and implementation of the SIMT-PS on a GPU computing platform. Section 5 presents computational experiment results and analysis. Conclusions and future research tasks are summarized in Section 6.

2 GPU Computing

GPU computing is an exciting new computing environment that is fundamentally different than the traditional CPU environment. A GPU is designed to process thousands of threads simultaneously enabling high computational throughput across large amounts of data. This research selects the Compute Unified Device Architecture (CUDA) technology from nVidia^{TM} to implement our algorithm. The CUDA environment allows a software developer to program a GPU for general purpose computing in a C programming environment [8].

The CUDA environment is designed to run thousands of threads concurrently with a same instruction set in a data parallel manner. Each thread runs an instruction set called a 'kernel'. Developers in a GPU environment have several different memory locations to store data. A kernel can employ 'registers' as fast access memory. The communication among threads can be realized with 'shared memory', which is a type of very fast memory that allows both read and write access during kernel run time. However, during a Pattern Search, all the searches are completed independently. Hence there is no need to use any 'shared memory' to exchange the information between threads during run time.

The communication between CPU and GPU can be done through global device memory, constant memory, or texture memory on a GPU board. Global device memory is a relatively slow memory location that allows both read and write operations. Texture memory is relatively fast memory that is read-only. Constant memory is fast read-only memory whose size cannot be dynamically changed in runtime. An investigation of the Pattern Search reveals that there is no read-only data during the searches, as the data are constantly updated. Therefore, texture memory and constant memory are not used in our implementation of the SIMT-PS algorithm. The nVidia GeForce GTX 280 GPU hardware employed in this paper has 30 multi-processors. Each multi-processor has 8 processors (or cores). This amounts to 240 data-parallel processors (cores) on one GPU board. Each multi-processor has 16K shared memory, 16K registers, 64K constant memory, and access to 1GB global device memory and texture memory for larger data storage [7]. Due to the different performance of memory locations and the limited amount of fast and flexible memory locations, algorithm implementation and design choices must be guided by memory limitations.

After compilation by the CUDA environment, a program runs as a kernel in a GPU. A kernel takes input parameters, conducts computations, and outputs the results to device memory where the result can be read by the CPU. Each thread must perform the same operation in the kernel, but the input data can be different. The CPU owns the host code that prepares input data and accepts output values from the GPU. The CPU is also responsible for reading and writing data files, storing solution values, and managing threads. The output data from the GPU is written to global device memory to be retrieved by a CPU program. Also, code with excessive conditional branching should also be avoided in a GPU environment since all threads execute the same instructions. The communication between the CPU and GPU requires some overhead time. To compensate for the overhead time, developers must send tasks to the GPU that are significantly larger than the communication overhead.

In our SIMT-PS algorithm, the initial solutions are generated by the CPU and then passed to GPU for Pattern Search. The solutions are constantly changing during the Pattern Search procedure. Hence, the solution data must be kept in the Global Device Memory. To minimize the impact of repetitive read and write access with the Device Memory, the solution array data are organized into a coalesced structure. At the end of Pattern Search, the solution data are copied back to the CPU host memory for further processing. The CPU-GPU communication is thus minimized. The computational experiments results also suggest that a multi-walk Pattern Search is an appropriate design for the GPU computing platform due to its low memory requirement and simple repetitive tasks.

3 Pattern Search

The basic Pattern Search algorithm is a direct search method that does not require derivative or second derivative information [9]. While this robustness makes Pattern Search a safe choice for solving many different types of problems [1], this research selects Pattern Search due to the memory limitation and the desire to avoid excessive branching in GPU hardware. The basic PS algorithm moves along the coordinate axes or other user defined positive spanning set to improve an existing solution in a greedy way. The step size Δ is reduced, typically in half, when an improvement is not found for any direction.

The feature of Pattern Search makes it amenable to parallel computing as it involves frequent objective function evaluations. For a complex objective function, this repetitive function evaluation computation task becomes very heavy and thus makes the Pattern Search a questionable selection as a direct search method on a single CPU platform. Parallel computing can alleviate this problem. Parallel Pattern Search methods have been proposed in the past research in CPU platforms [2], [3]. In a GPU parallel computing platform, the benefit can be much more evident, as GPU computing does not have the large overhead of a CPU cluster which often seriously reduces the benefit of parallel computing. Therefore, the savings on the computation time of Pattern Search on a GPU computing platform can be tremendous.

Alternatives to Pattern Search include Nelder-Mead simplex search method and gradient based methods [5]. The Nelder-Mead simplex search requires data storage for $d+1$ solution vectors each of size d where d is the number of variables in the problem. Gradient methods based methods require $O(d^2)$ memory. The Pattern Search method requires a single solution vector with d elements, the cost of the prior solution, and the current step sizes for $d+3$ memory elements per search thread. For current GPU hardware and our algorithm design, a number of 15360 parallel threads achieved the best speedup performance, as presented in Section 5. For example, with a 100 variable problem, single precision storage, and 15360 threads, Pattern Search requires 6 MB of memory compared to over 600 MB for a gradient based method and Nelder-Mead simplex search method. Given the current limitation in memory in GPU hardware, this difference in memory requirement is a tremendous advantage for Pattern Search over alternative methods that require more memory. In addition, this small memory consumption makes it possible for Pattern Search to work as a sub-component in a bigger problem-solving framework.

4 'Single Instruction – Multiple Threads – Pattern Search' (SIMT-PS) Algorithm

Pattern Search is modified for the GPU SIMT computing environment by conducting multiple searches in parallel and searching a fixed number of iterations instead of a limit based on solution tolerance. Each thread represents one independent search. Let x denote the solution vector of a particular thread composed of d elements. The initial values of x are set randomly based on a uniform distribution from the lower bound to the upper bound.

$$x_i = \text{Uniform}[a_i, b_i] \quad i = 1, 2\ldots, d \tag{1}$$

From this initial seed point, we improve each thread using a Pattern Search algorithm.

Our pattern denoted by $\mathsf{D} \equiv \{e_1^+, e_1^-, \ldots, e_d^+, e_d^-\}$ is defined by the unit coordinate axes where d is the dimension of the problem where e_j is the standard unit base vectors. The PS is initialized by setting the step size Δ_i to a user specified initial Δ_0. The PS in this paper explores the coordinate axes for improving solution for k iterations. In general, the pattern can be user defined positive spanning set but the coordinate axis is the easiest to implement. If the search does not find an improvement for *any direction*, then the step size Δ_i is reduced by half. A classical PS is typically stopped once the step size is less than a user specified tolerance, but in our algorithm, the PS is stopped after a fixed number of iterations. With a large number of threads in a SIMT computing environment, waiting until all threads reach convergence is not an effective use of computing resources. If a thread reaches convergence to a user specified tolerance Δ_{tol} before the iteration limit reached, this research resets the step size Δ_i to the initial Δ_o to make use of computer resources that otherwise would be idle. Fig. 1 presents the steps of the algorithm. All threads are initialized to random starting solutions for the solution vector x and step sizes of Δ_o. For k_{max} iterations the algorithm performs a Pattern Search operation on all threads for each variable of the problem. If no improvement is found for any variable during an iteration, the step size is reduced by half. Finally, the best solution is selected from all of the threads (Fig. 1).

```
For all threads (done in GPU)
    a) Initialize the step size to $\Delta_o$ and the solution $x_i$ to a random starting solution where $x_i$=
       Uniform[$a_i$, $b_i$];
    // loop until max iteration
    For (iteration $k = 0 \ldots k_{max}$)
      For (dimension $j = 1 \ldots d$)
        b)If $f(x_i + e_j^+\Delta_i) \le f(x_i)$ and $f(x_i + e_j^+\Delta_i) \le f(x_i + e_i^-\Delta_i)$ then $x_i = x_i + e_j^+\Delta_i$.
        c)Else If $f(x_i + e_j^-\Delta_i) \le f(x_i)$ and $f(x_i + e_j^-\Delta_i) \le f(x_i + e_i^+\Delta_i)$ then $x_i = x_i + e_j^-\Delta_i$.
      End For (each dimension)
    d) If no improvement throughout all $d$ dimensions, $\Delta_i = \Delta_i / 2$; If $\Delta_i = \Delta_{tol}$, reset $\Delta_i = \Delta_o$;
    End For (k iterations)
End For (threads)
    e) Select best thread based on solution cost (done in CPU)
```

Fig. 1. Procedure of the SIMT-Pattern Search algorithm

5 Computational Results and Analysis

The proposed Single Instruction Multiple Thread –Pattern Search (SIMT-PS) algorithm for the general optimization functions has been implemented in Visual C++ 2005 environment with the CUDA environment for programming the GPU. The computational experiments were executed on a Dell Precision 7400 Workstation computer with an Intel® Core™ 2 Duo 2.5GHz CPU, 3GB memory, and an nVidia GeForce™ GTX 280 GPU. For benchmarking, the algorithm was also implemented with CPU-based only functions to compare the computation speed to the GPU-accelerated implementation. The same computer was used in testing both CPU and GPU versions. The GPU version of the algorithm requires many threads to unleash its full potential. The peak speedup performance is expected when the number of threads is sufficient to keep the multi-processors busy at the same time. The GeForce GTX 280 GPU employed in this research has 30 multi-processors. Due to register constraints, each multi-processor can support 2 blocks with each block having 256 threads. In our hardware, 15360 (30 multi-processors * 2 blocks per multi-processor * 256 threads per block) provides good performance per thread. As anticipated by this analysis, Table 1 shows that the GPU code reaches peak performance relative to the CPU code when the number of threads is 15360.

Twelve benchmark functions have been selected for these computational experiments [10]. These benchmark functions are listed in the Appendix A. The Δ_o is set to be equal to (parameter range / 20). During the search process, Δ can be reduced to as small as Δ_{tol}, which is set to be $\Delta_o/2^{16}$.

We hope to first gain better understanding of how and where the SIMT-PS improve over its corresponding CPU implementation. To attain this goal, we picked the first benchmark function in the Appendix A, the Ackley function, for this investigation. Table 1 shows a comparison of computation times between the proposed SIMT-PS and CPU implementation of the same algorithm. Both software implementations execute the same search to ensure a fair comparison. The tests were conducted on Ackley function with dimension 20 (i.e., 20 variables). For the SIMT-PS algorithm, the recorded time is combined CPU and GPU code running time, including initialization overhead. As shown in Table 1, the computation time increases as the number of threads (parallel searches) increases. As the number of threads increases, the GPU codes performance per thread improves whereas the CPU performance remains relatively constant. The speedup values are calculated by dividing the CPU PS algorithm times by the SIMT-PS algorithm times.

Table 1. Comparison of computation times between the SIMT-PS and the PS algorithms as function of thread numbers (Ackley Function, 20 variables, 20 PS iterations, average of 10 Monte Carlo Runs, time shown in milliseconds)

Threads	GPU Code	CPU Code	Speedup
2048	21.9	1267.1	57.86
4096	29.7	2543.7	85.65
8192	51.6	5098.4	98.81
12288	64.1	7629.6	119.03
15360	76.6	9535.9	124.49

To understand the performance improvement on specific computational tasks within the algorithm, we examined the run time of different portions of the code in both the GPU and CPU algorithm implementations. As can be seen from Table 2, Pattern Search on the GPU platform significantly reduced the computation time for the search process. The 'Other functions' listed in the Table 2 include problem initialization, initial objective function evaluations and summary computation. The 'Select Best Thread' operations in both versions are completed with CPU code, but in the SIMT-PS algorithm, additional time is needed for memory manipulation.

Table 2. CPU and GPU time breakdowns for each task and their comparison (15360 threads, 20 variables, 20 PS iterations, time shown in milliseconds)

Task	GPU Code	CPU Code	Time %
1) Other functions	33.81	46.60	72.5%
2) Select Best Thread	6.57	4.65	141.5%
3) Pattern Search	35.73	9446.68	0.4%

Two concerns when designing GPU programs are CPU overhead for preparing for GPU tasks and the communication overhead between the CPU and the CPU. We conduct several tests with different numbers of threads to determine the times spent on each GPU task. The results are presented in Table 3. These tests are conducted with the assistance of *CUDA Visual Profiler* software, a tool for analyzing GPU kernel performance, available for download from nVidia website [7]. The 'GPU' column is the total time on GPU in executing the task. The 'CPU' column is the overhead time needed in CPU in order to run the corresponding GPU task. The '% GPU time' column is found by dividing each GPU task times by the total time spent in the GPU. The actual time spent on each GPU task is the sum of the 'GPU' and 'CPU' columns. The results show that the Pattern Search kernel take the majority of the time spent on GPU. The CPU-GPU memory synchronization takes a very small portion of the GPU time. The results also show that the CPU overhead is reasonably small in our implementation. Tracking CPU and communication overhead is an important performance tuning tool when developing GPU-enabled algorithms.

Table 3. GPU time and corresponding CPU overhead time used on each GPU task (15360 threads, 20 variables, for a short run, time shown in μ-seconds)

GPU Task	Calls	GPU μsec	CPU μsec	Percent of Time
Pattern Search	21	729850	110	98.5%
Initial Fitness Functions	22	1138	199	0.2%
Memory Synchronization	107	9717		1.3%

As shown in Table 4, the performance of the GPU version of the code is significantly faster than the CPU version of the code on 12 test problems as listed in the Appendix A. The speedup ranged from 30 to 138. The performance improvement is generally greater on problems where the objective function requires more time to evaluate and the overall solution time is longer.

Time to solution is a critical performance measure for an optimization procedure. Table 5 shows the comparison between the SIMT-PS and the PS with a run time limit of 1 second. Since the SIMT-PS algorithm is very fast (typically finishes within 100 milliseconds for a 30 variable and 20 iteration Pattern Search), we make it run repetitively from different initial solutions once one round of parallel Pattern Search is over. Given enough time, both CPU and GPU versions of our implementation should give satisfactory results. When time is limited, the result obtained with GPU-accelerated SIMT-PS algorithm is better, as it has time to search wider search space. The optimal solutions of all test problems are 0. In fact, the CPU PS implementation for some test functions cannot finish one round of Pattern Search within 10 seconds, but we let the CPU code run to finish at least one Pattern Search. The actual time spent by each algorithm is also listed in Table 5.

Table 4. Comparison of Computation Times between the SIMT- PS and its corresponding CPU Implementations on all 12 test functions (15360 threads, 20 variables, 20 PS iterations, time shown in milliseconds)

#	Problems	CPU	GPU	Speedup
1	Ackley	9523.4	78.2	121.8
2	Griewank	9675	92.2	104.9
3	Penalty1	63537.5	484.3	131.2
4	Penalty2	61865.6	509.4	121.4
5	Quartic	2375	51.6	46.0
6	Rastrigin	9465.6	70.3	134.6
7	Rosenbrock	6421.9	70.3	91.3
8	Schwefel 1.2	15353.1	117.2	131.0
9	Schwefel 2.22	1890.7	50	37.8
10	Schwefel 2.21	2120.3	46.9	45.2
11	Sphere	1393.7	45.3	30.8
12	Step	6943.8	50	138.9

Table 5. Time limited results comparison between the SIMT-PS and its corresponding CPU Implementation (10 Monte Carlo runs, 15360 threads, 30 variables, 20 PS iterations, max 10 seconds, time shown in milliseconds)

#	Problems	Best Solution		Mean Best Solution		Actual Time Spent	
		CPU PS	SIMT-PS	CPU PS	SIMT-PS	CPU PS	SIMT-PS
1	Ackley	0.000	0.000	0.000	0.000	19,907	10,074
2	Griewank	0.064	0.004	0.126	0.007	21,271	10,070
3	Penalty1	0.000	0.000	0.001	0.000	140,970	10,041
4	Penalty2	0.008	0.000	0.021	0.000	136,574	10,374
5	Quartic	0.000	0.000	0.000	0.000	10,131	10,066
6	Rastrigin	54.588	18.196	73.593	24.666	21,036	10,076
7	Rosenbrock	22.340	1.020	25.946	13.146	14,430	10,078
8	Schwefel 1.2	4,417.340	279.460	5,036.064	539.862	46,755	10,251
9	Schwefel 2.22	0.000	0.000	0.000	0.000	13,441	10,042
10	Schwefel 2.21	12.900	3.200	17.660	3.510	12,494	10,027
11	Sphere	0.000	0.000	0.000	0.000	11,708	10,045
12	Step	0.000	0.000	0.000	0.000	15,142	10,030

6 Conclusions and Future Research

On a GPU computing platform, Pattern Search is an effective optimization method for nonlinear bound constrained optimization due to its small memory requirement and fast parallel objective function evaluations. Using GPU hardware can generally speed up Pattern Search by a factor of 100 compared to CPU implementations. On a hardware cost per computing operation comparison, GPU computing is currently attractive for optimization heuristics. Research and practitioners who solve optimization problems with heuristics always benefit from low cost computing resources. Improved computing resources can be deployed to solve problems quicker or to find better solutions. This low cost platform should encourage heuristic designers to develop algorithms that are effective in a data parallel, low memory environment. The approach taken in this paper of using a simple search procedure with a massive number of search threads is a promising approach for applying GPU technology to optimization problems. As future research, computational studies exploring the performance of alternative heuristics in low memory data parallel environments would aid the adoption of GPU technology to optimization problems.

Acknowledgments. This work was partially supported by the Lamar Research Enhancement Grants to Dr. W. Zhu and Dr. J. Curry at Lamar University. Partial support for this work was provided by a grant from the Texas Hazardous Waste Research Center to Dr. W. Zhu, Dr. J. Curry and Dr. H. Lou at Lamar University. Partial support for this work was provided by the National Science Foundation's Award No. 0737173 to Dr. W. Zhu. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF. Their support is greatly appreciated.

References

1. Audet, C., Dennis Jr., J.E.: Analysis of Generalized Pattern Searches. SIAM Journal of Optimization 13, 889–903 (2003)
2. Hough, P.D., Kolda, T.G., Torczon, V.J.: Asynchronous Parallel Pattern Search for Nonlinear Optimization, SAND2000-8213, Sandia Lab Reports (January 2000)
3. Kolda, T.G.: Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization. SIAM Journal of Optimization 16, 563–586 (2005)
4. Lewis, R.M., Torczon, V.J.: Why Pattern Search Works, NASA/CR-1998-208966, ICASE Report No. 98-57 (1998)
5. Lewis, R.M., Torczon, V.J., Trosset, M.W.: Direct Search Methods: Then and Now. Journal of Computational and Applied Mathematics 124(1-2), 191–207 (2000)
6. Li, J., Wan, D., Chi, Z., Hu, X.: A Parallel Particle Swarm Optimization Algorithm based on Fine-grained Model with GPU-Accelerating. Journal of Harbin Institute of Technology 38, 2162–2166 (2006)
7. nVidia: CUDA Programming Guide V 2.0 (2008), http://www.nvidia.com/object/cuda_get.html
8. Nguyen, H. (ed.): GPU Gems 3. Addison-Wesley, New York (2007)
9. Torczon, V.J.: On the Convergence of Pattern Search Algorithms. SIAM Journal of Optimization 7, 1–25 (1997)
10. Yao, X., Liu, Y., Lin, G.: Evolutionary Programming Made Faster. IEEE Transactions on Evolutionary Computation 3, 82–102 (1999)
11. Zhu, W., Curry, J., Marquez, A.: SIMD Tabu Search with Graphics Hardware Acceleration on the Quadratic Assignment Problem. Accepted by the International Journal of Production Research (2008)

Appendix A – Test Functions

Due to the page limit, only two test functions are described here. Other functions definition can be found in reference [10].

1) Ackley Function:

$$f_1(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}, \quad -30 \le x_i \le 30, \quad i = 1,2,...,n$$

$\min(f_1) = f_1(0,...,0) = 0$

2) Griewank Function

$$f_2(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right), \quad -600 \le x_i \le 600, \quad i = 1,2,...,n$$

$\min(f_2) = f_2(0,...,0) = 0$

3) Generalized Penalized Function 1
4) Generalized Penalized Function 2
5) Quartic Function
6) Rastrigin Function
7) Rosenbrock Function
8) Schwefel's Problem 1.2
9) Schwefel's Problem 2.22
10) Schwefel's Problem 2.21
11) Sphere Function
12) Step Function

A Massively Parallel Architecture for Bioinformatics

Gerd Pfeiffer[1], Stefan Baumgart[1], Jan Schröder[2], and Manfred Schimmler[2]

[1] SciEngines GmbH, 24118 Kiel, Germany
http://www.sciengines.com
[2] Christian-Albrechts Universität, Department of Computer Science,
Hermann-Rodewald-Str. 3, 24118 Kiel, Germany
http://www.informatik.uni-kiel.de

Abstract. Today's general purpose computers lack in meeting the requirements on computing performance for standard applications in bioinformatics like DNA sequence alignment, error correction for assembly, or TFBS finding. The size of DNA sequence databases doubles twice a year. On the other hand the advance in computing performance per unit cost only doubles every 2 years. Hence, ingenious approaches have been developed for putting this discrepancy in perspective by use of special purpose computing architectures like ASICs, GPUs, multicore CPUs or CPU Clusters. These approaches suffer either from being too application specific (ASIC and GPU) or too general (CPU-Cluster and multicore CPUs). An alternative is the FPGA, which outperforms the solutions mentioned above in case of bioinformatic applications with respect to cost and power efficiency, flexibility and communication bandwidths. For making maximal use of the advantages, a new massively parallel architecture consisting of low-cost FPGAs is presented.

1 Introduction

Bioinformatics algorithms are most demanding in scientific computing. Most of the times they are not NP-hard or even of high asymptotic complexity but the sheer masses of input data make their computation laborious. Furthermore, the life science thus bioinformatics research field is growing fast and so is the input data that wait to be processed. Besides the massive amount of data to be processed, the field of bioinformatics algorithms have another characteristic in common: simplicity. Algorithms for the big areas of bioinformatics like sequence alignment, motif finding, and genome assembly that are used all around the world are impressively simple in their computation. We refer to Smith, Waterman [9], Needleman, Wunsch [10], or BLAST [11] as examples for simple yet effective and popular sequence alignment algorithms; expectation maximization (EM/MEME) [12], projection algorithm [13] for motif finding; EulerSR [8] and Edena [7] for genome assembly. All these programs use only very simple but repetitive operations on the input data: very simple arithmetic operations, string matching and comparison.

Standard CPUs are designed for providing a good instruction mix for almost all commonly used algorithms [2]. Therefore, for a class of target algorithms they can not be as effective as possible in terms of the design freedom. The same with the described purpose: here the arithmetic requirement is too simple to demand a full CPU and the string matching does not fit well at all to the hardware structure of a CPU. Hence many clock cycles will be consumed even for simple comparisons of nucleotides. Also, most of the algorithms above have a trivial approach to parallelization inherent another characteristic that does not fit too well with standard CPUs. The result of this are high runtimes or necessity for PC clusters, massive consumption of computer memory, and not to forget a vast amount of energy used - thus bare money to be spent.

This motivates the use of special purpose hardware. Several approaches exist to introduce special hardware to bioinformatics: Special ASIC [16], GPUs [17] and FPGAs [19] [20] [21] for example. FPGAs are to be named in particular because they can be adjusted exactly to the demands of an application like an ASIC and yet being able to readjust if the application changes slightly or even completely which is a powerful ability in a fast changing field like bioinformatics with a lot of different applications as mentioned above [1]. In this article we present a massively parallel special purpose hardware based on low cost FPGAs called COPACOBANA 5000. Its architecture and name is based on a similar machine presented earlier. It is the Cost-Optimized PArallel COde Braker ANd Analyzer (COPACOBANA) 1000 [4] The original purpose of this machine was cryptanalysis: fast code breaking of the DES standard and some further attacks [3]. It also proved to be applicable for bioinformatics though, allowing speedups of a factor 10,000 at a fraction of the energy costs in motif finding application compared to a single PC [14] [15].

The new architecture presented here makes use of the high applicability of FPGA chips in bioinformatics. The design goal is gaining a maximal speedup by maximizing the number of chips working together at low hardware costs and low energy consumption. The connection of the chips is realized by a sophisticated bus system allowing as much throughput as possible in order to be able to deal with the huge amounts of input data.

This paper is organized as follows: In Section 2 a first approach to massive parallelization with FPGAs is shown. Section 3 describes the new architecture allowing higher data throughput and bigger FPGA chips to challenge the big problems of bioinformatics in short time. In Section 4 the applicability and the estimated speedups gained and energy saved by this architecture are shown. Section 5 concludes the paper.

2 Copacobana 1000

COPACOBANA 1000 (Fig. 1) is a massively parallel reconfigurable architecture consisting of 120 low-cost FPGAs. Its hardware architecture has been developed according to the following design criteria: First, it was assumed that computationally costly operations are trivial parallelizable in terms of interprocess

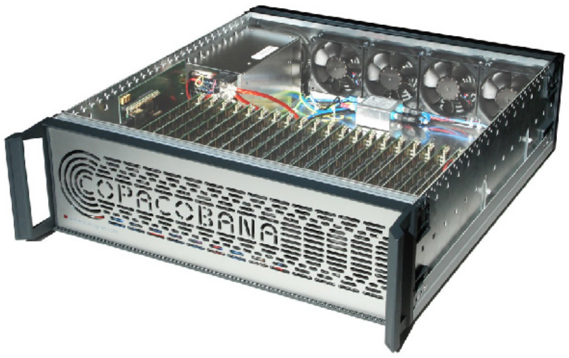


Fig. 1. The Copacobana 1000 machine

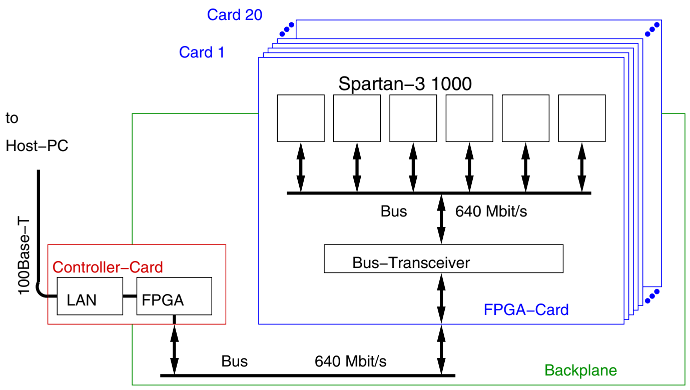


Fig. 2. The Copacobana 1000 architecture

communication requirements. Second, the demand for data transfer between host and nodes is low due to the fact that computations heavily dominate communication requirements. A communication of low bandwidth between the parallel machine and a host computer is transferring instructions and results. Hence, a single conventional computer as front end is sufficient to transfer the required data packets to and from the acceleration hardware. The COPACOBANA is connected by a local area network (LAN) interface. Third, all target algorithms and their corresponding hardware nodes demand for very little local memory, which can be provided by the on-chip RAM modules of an FPGA.

Since the cryptanalytical applications demand for plenty of computing power, a total of 120 FPGA devices on the COPACOBANA cluster have been installed. By stripping down the hardware functionality of COPACOBANA to the bare minimum, an optimal cost-performance ratio for code breaking has been achieved. For the optimal cost-performance ratio, the system was designed on basis of small FPGA chips which come with a high ratio of logic ressources per cost. Further optimization on system level lead to a partitioning into 20 subunits that can be dynamically plugged into a backplane. Fig. 2 illustrates this architecture. Each of these modules in DIMM form factor hosts six low-cost Xilinx Spartan-3 XC3S1000 FPGAs that are directly connected to a common 64-bit data bus on-board. The data bus of the module is interfaced to the global data bus on a backplane. While disconnected from the global bus, the FPGAs on the same module can communicate via the local 64-bit data bus. Additionally, control signals are run over a separate 16-bit address bus. For simplicity, a single master bus was selected to avoid interrupt handling. Hence, if the communication scheduling of an application is unknown in advance, the bus master will need to poll the FPGAs. The front end of COPACOBANA is a host PC that is used to initialize and control the FPGAs, as well as for accumulation of results. Programming can be done in different levels for all or one specific subset of FPGAs. Data transfer between FPGAs and a host PC is accomplished by a dedicated control interface. This controller has also been designed as a slot-in module so that COPACOBANA can be connected to a computer either via a USB or Ethernet controller card. A software library on the host PC provides low-level functions that allow for device programming, addressing individual FPGAs, and storing and reading FPGA-specific application data. With this approach, more than one COPACOBANA device can easily be attached to a single host PC.

3 Copacobana 5000

The new architecture COPACOBANA 5000 consists of an 18 slot backplane equipped with 16 FPGA-cards and 2 controller cards. The latter connect the massively parallel FPGA-computer to an in-system off-the-shelf PC. Each of the FPGA-cards carry 8 high performance FPGAs interconnected in a one dimensional array. Additional units are supporting the mentioned functional units as for example a $1.5kW$ [1] main power supply unit, 6 high-performance fans and a 19-inch rack of 3 hight units (3HE) for the housing.

3.1 Bus Concept

The interconnection between the individual FPGA-cards and between the FPGA-cards and the controller is organized as a systolic chain:

There are fast point-to-point connections between every two neighbors in this chain. The first controller communicates with the first FPGA on the first card and the last FPGA on the last card is connected to the second controller. To speed up global broadcasts there are possible shortcuts in the chain between adjacent FPGA-cards. The point-to-point interconnections consist of 8 pairs of wires in each direction. Each pair is driven by low voltage differential signalling

[1] On output site: 125A + 12V.

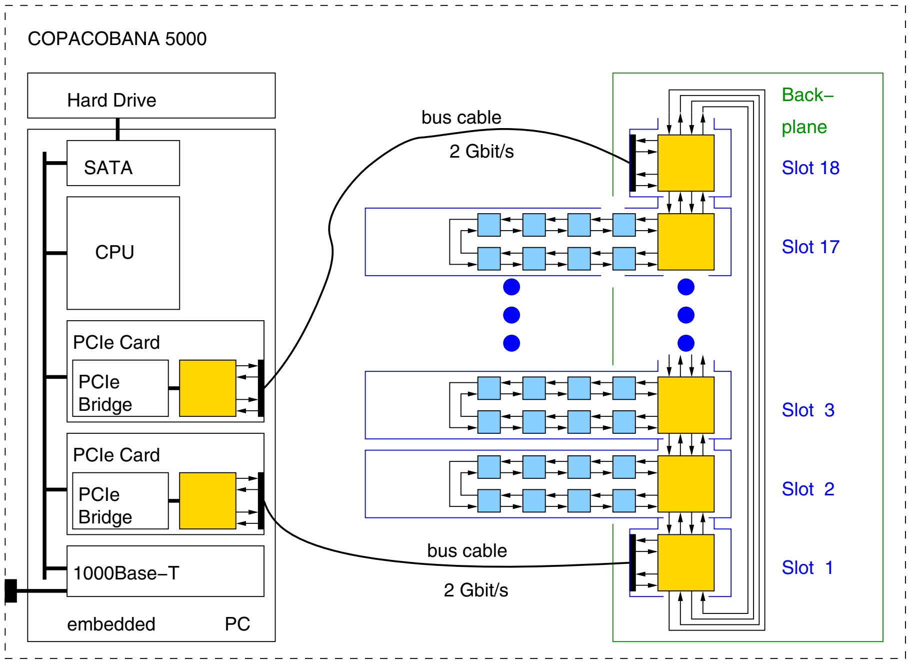


Fig. 3. The Copacobana 5000 System

(LVDS) with a speed of 250MHz, thus achieving a data-rate of 2Gbit/s. Figure 3 shows the overall architecture of COPACOBANA 5000.

3.2 Controller

The root entity of control is running on a remote host computer. This machine is integrated into a local area network (LAN) which allows to connect to the COPACOBANA 5000. Two scenarios are considered here. The first is that the database is physically located inside the COPACOBANA machine e.g. on an SATA hard drive attached to the embedded PC. In this case the remote computer functions as user terminal only. The second scenario is that the embedded PC is used for an easy access to standard interfaces only. Here 2 Gigabit Ethernet LAN ports which is standard for recent main boards are of interest for the second scenario. The database has to be transfered over the LAN. In any case the remote host computer is initiator of every activity. The next instance is the embedded PC. Here the control information is translated for COPACOBANA and dropped into the system. In most cases it is not required for the higher level control instance to check if a command is executed successfully or not in the lower level of control hierarchy. However, it is possible if demanded by the implemented algorithm for example.

One board is plugged into a PCIe interface connector of the embedded PC. On this card an FPGA transceives the data and control by a proprietary wire

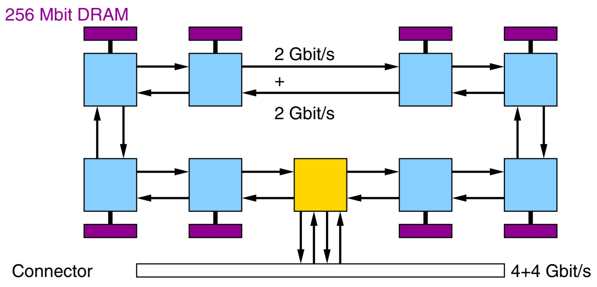


Fig. 4. Data Path of the Copacobana 5000 FPGA-Card

connection to the second half of the controller. Here another FPGA decodes the incoming data and instructions and drops them into the bus system of COPACOBANA 5000.

3.3 FPGA-Card

The FPGA-card is depicted in Fig. 4. It consists of 8 FPGAs of the type Xilinx Spartan-3 5000 for running the application and an additional FPGA which comes with a fixed configuration. The latter is routing the systolic datastreams. This simple protocol provides address information in header fields of the data stream. All FPGAs are globally clocked synchronously. Each clock cycle the data are transferred from one FPGA to the adjacent one building a huge communication pipeline. Inside the system this systolic data flow allows a throughput of 2Mbit/s for each direction. The protocol acquires an overhead of 20%. Between the controller-cards and the embedded PC the maximum data-rate is limitid to 250MByte/s due to the PCIe connection. The dissadvantage of the high throuput is a considerable amout of latency depending on the path the data are travelling.

3.4 Backplane

The backplane is holding the plugged cards mechanically, generating and distributing the clock signals, distribution the power, and finally connecting the cards for communication. The FPGA-card can transfer the incoming data to the next slot or it can take the data out of the stream. In this case an empty data field will go systolically through the bus pipeline from slot to slot. Another card can insert new data into this empty slot. The two counterrotating systolic dataflows allow to minimize the worst case latency to half of the total number of slot times the clock cycle time. But instead of a single master shared bus, the new machine connects one ascending slot and one descending slot to each single card. This leads to a ring of point to point connections. As the system is globally clocked the incoming data are registered. As result the bus system is working similar to a parallel shift register.

3.5 Memory

The big problem of using FPGAs in bioinformatics is the memory consumption. As mentioned in the introduction, the volume of input data is huge. FPGAs on the contrary are rather small in memory capacity. Our approach to solve this disparity is to use the connecting bus as a kind of memory replacement for the input data. Most of the mentioned algorithms have in common that they use only a small portion of memory (kilobytes) to store intermediate and end results. On the other hand they have to access huge amounts of data (gigabytes) to generate these results. The latter data needed for the application will be provided on the bus so every chip has access to the input data without storing it on chip site. However, some algorithms demand quite a lot of memory for the intermediate results as well some sequence alignment or assembly algorithms for example. To be able to handle those algorithms too (for a decent set of input parameters) we provide external local memory located next to each FPGA chip to guarantee applicability for most purposes.

In order to avoid a bottleneck, the raw input data have to pass the system with high throughput. Fortunately the target algorithms are latency insensitive due to the high locality of the parallelized bioinformatics algorithms. For ensuring a very high throughput in the design of the bus system, the I/O capabilities of the FPGA has to be analyzed carefully. The highest throughput can be achieved by connecting communicating chips by point-to-point lines of a short length [5] [6].

In addition, each FPGA is equipped with some external local memory. There is a 32MByte chip connected to each individual Spartan-3 5000 chip.

3.6 Software

Communication with COPACOBANA 5000 follows the principle of Memory-mapped-I/O (MMIO). By this a host software writes and reads data to and from addressed FPGAs, and inside to addressed IO-registeres. The latter are application specifically used for data or control words. The control words are commonly incooperated into finite state machines. A communication framework builds a bridge between the two user programmed entities, the host software and the user FPGA core. Both are connected by a framework interface. At host site an Application Programming Interface (API) provides easy access to the machine by a set of communication library calls for the MMIO commands. The API supports Java and C++. On the other side of the framework interface an Relationally Placed Macro (RPM) has to be embedded into the user FPGA configuration. One side of this macro connects to the physical bus system via IO-blocks and inside the FPGA the macro connects to the user implementation as API for FPGA design, for example in VHDL. The common communication principle of Memory-mapped-I/O is easy to understand by the user. Together with the given communication framework a user does not need to know the COPACOBANA 5000 in detail. However, the knowledge about the architecture is helpful for implementing parallel algorithms efficiently.

Table 1. COPACOBANA Performance in Exhaustive Key Search on DES

	Intel Pentium-4 3.0GHz	Copacobana 1000	Copacobana 5000 (est.)
encryption speed	$2 \cdot 10^6 \frac{1}{s}$	$6.5 \cdot 10^{10} \frac{1}{s}$	$3.0 \cdot 10^{11} \frac{1}{s}$
average search time	$571 years$	$6d10h$	$1d10h$
energy consumption	$750MWh$	$93kWh$	$62kWh$

Table 2. COPACOBANA Performance in Motif-Finding

	Intel Xeon 5150 2.6GHz dual core	Copacobana 1000	Copacobana 5000 (est.)
Cowpox Virus (230kbp)	$144h$ $21.6kWh$	$1h40m$ $1.0kWh$	$21m$ $0.63kWh$
Rickettsia canadensis (1.2Mbp)	$2,575h$ $386.2kWh$	$4h10m$ $2.5kWh$	$52m$ $1.6kWh$
Bacillus subtilis (5.1Mbp)	$11,236h$ $1685.4kWh$	$16h45m$ $10.1kWh$	$3h30m$ $6.3kWh$

kbp : kilo base pairs, Mbp : Mega base pairs.

4 Performance Estimation

In this section the performance is compared between a PC, a COPACOBANA 1000 and the estimated performance of the COPACOBANA 5000.

The COPACOBANA 1000 originally was intended for running cryptanalytical applications. In Table 1 the performance of an exhaustive key search on DES is compared between an Intel Pentium-4 3.0GHz and the implementation on the COPACOBANA 1000. The estimated values for the COPACOBANA 5000 are simply based on the number of FPGAs and the size of the chip. Each Spartan-3 5000 comes with 4.5 times of the logic ressources compared to the Spartan-3 1000. Furthermore the new machine hosts 128 user FPGAs and the old one 120. Hence, the new machine has approximately 4.8 times more computing performance. This assumption is legitimately due to fact that both chips are based on the same technology and are different in the size only.

Despite that fact, a proof of concept for the applicability in bioinformatics has been developed. In Table 2 the results of this research are shown. The target application is motif finding on DNA sequences. The implementation on the COPACOBANA 1000 has been tested and the performance compared to a standard computer has been measured. The estimation of the performance of the COPACOBANA 5000 is based on the number and size of FPGAs as explained above. Probably the performance will exceed these values due to the optimized bus.

One of the most interesting aspects is the one of power consumption. COPACOBANA 1000 and COPACOBANA 5000 are extremely power efficient for the considered applications. Observe that already a single run of one exhaustive key search on the Data Encryption Standard (DES) by use of PCs [2] saves costs of power consumption in the scale of the purchase costs of a COPACOBANA 5000.

5 Conclusion

Bioinformatics applications are computationally extremely demanding. It is fair to believe that analyzing biological research data as for example DNA sequence data with conventional PCs and super computers is far too expensive. The only promising way to tackle existing computing machines is to build special purpose hardware, dedicated solely to suitable algorithms such as those presented in this paper.

Conventional parallel architectures turn out to be far too complex and, thus, are not cost efficient in solving bioinformatics problems. Most of these problems can be parallelized easily and we show the architecture of the recent design the COPACOBANA 5000 which results from the algorithmic requirements of the targeted cryptanalytic problems.

Recapitulating, the COPACOBANA machines are the first and currently the only available cost efficient massively parallel FPGA-computers . The COPACOBANA 1000 was intended to, but is not necessarily restricted to solving problems related to cryptanalysis. A proof of concept confirmend the applicability for high-performance bioinformatics computing.

The work at hand presents the design and architecture of a cost efficient advancement of the old design fulfilling the request for bioinformatics computing. The COPACOBANA 5000 will host 128 low-cost FPGAs. We showed, by extrapolating a successfully implemented proof of concept on the COPACOBANA 1000 that the new hardware architecture will reach increased performance by a factor of five compared to the old machine and better than a standard computer in orders of magnitude.

Future work includes completion and optimization of all performance relevant design issues. The implementations have to be optimized to guarantee best possible throughput. The first prototype of COPACOBANA 5000 is to be presented in May 2009.

References

1. DeHon, A.: The Density Advantage of Configurable Computing. IEEE Computer Magazine 33(4), 41–49 (2000)
2. Hennessy, J.L., Patterson, D.: Computer Architecture: A Quantitative Approach (1995)
3. Guüneysu, T., Kasper, T., Novotny, M., Paar, C., Rupp, A.: Cryptanalysis with COPACOBANA. Transactions on Computers 57, 1498–1513 (2008)

[2] Cluster of Intel Pentium-4 3.0GHz.

4. Kumar, S., Paar, C., Pelzl, J., Pfeiffer, G., Schimmler, M.: COPACOBANA - A Cost-Optimized Special-Purpose Hardware for Code-Breaking. In: IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM 2006, Napa, California, April 24-26 (2006)
5. Montrose, M.I.: EMC and the Printed Circuit Board. IEEE Press, New York (1999)
6. Montrose, M.I.: Printed Circuit Board Design Techniques for EMC Compliance, 2nd edn. IEEE Press, New York (2000)
7. Hernandez, et al.: De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. Genome Res. 18(5), 802–809 (2008)
8. Chaisson, P.: Short read fragment assembly of bacterial genomes. Genome Research (December 2007)
9. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147, 195–197 (1981)
10. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48(3), 443–453 (1970)
11. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. Journal of Molecular Biology 215(3), 403–410 (1990)
12. Bailey, T.L., Elkan, C.: Unsupervised Learning of Multiple Motifs in Biopolymers Using Expectation Maximization. Techn. Report, Department of Computer Science and Engineering, University of California, San Diego, La Jolla, California, USA (1993)
13. Buhler, J., Tompa, M.: Finding Motifs Using Random Projections. Journal of Computational Biology 9(2), 225–242 (2002)
14. Schroeder, J., Schimmler, M., Tischer, K., Schroeder, H.: BMA Boolean Matrices as Model for Motif Kernels. In: 2008 International Conference on Bioinformatics, Computational Biology, Genomics and Chemoinformatics (2008)
15. Schroeder, J., Wienbrandt, L., Pfeiffer, G., Schimmler, M.: Massively Parallelized DNA Motif Search on the Reconfigurable Hardware Platform COPACOBANA. In: Chetty, M., Ngom, A., Ahmad, S. (eds.) PRIB 2008. LNCS (LNBI), vol. 5265, pp. 436–447. Springer, Heidelberg (2008)
16. Blas et al.: The UCSC Kestrel Parallel Processor. IEEE Transactions on Parallel and Distributed Systems 16(1) (January 2005)
17. Manavski, S.A., Valle, G.: CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. In: BMC Bioinformatics 2008, vol. 9(suppl. 2), p. 10, March 26 (2008)
18. Oliver, T.F., Schmidt, B., et al.: Multiple Sequence Alignment on an FPGA. ICPADS (2), 326–330 (2005)
19. Oliver, T., Schmidt, B., Maskell, D.: Reconfigurable Architectures for Bio-sequence Database Scanning on FPGAs. IEEE Transactions on Circuits and Systems II 52(12), 851–855 (2005)
20. Oliver, T., Schmidt, B., Nathan, D., Clemens, R., Maskell, D.: Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW. Bioinformatics 21(16), 3431–3432 (2005)
21. Oliver, T., Leow, Y.Y., Schmidt, B.: Integrating FPGA Acceleration into HMMer. Parallel Computing 34(11), 681–691 (2008)

GPU Accelerated RNA Folding Algorithm

Guillaume Rizk[1] and Dominique Lavenier[2]

[1] Univ-Rennes 1/IRISA
[2] ENS-Cachan/IRISA
IRISA - Symbiose Campus universitaire de Beaulieu,
35042 Rennes Cedex, France
{guillaume.rizk,dominique.lavenier}@irisa.fr

Abstract. Many bioinformatics studies require the analysis of RNA or DNA structures. More specifically, extensive work is done to elaborate efficient algorithms able to predict the 2-D folding structures of RNA or DNA sequences. However, the high computational complexity of the algorithms, combined with the rapid increase of genomic data, triggers the need of faster methods. Current approaches focus on parallelizing these algorithms on multiprocessor systems or on clusters, yielding to good performance but at a relatively high cost. Here, we explore the use of computer graphics hardware to speed up these algorithms which, theoretically, provide both high performance and low cost. We use the CUDA programming language to harness the power of NVIDIA graphic cards for general computation with a C-like environment. Performances on recent graphic cards achieve a $\times 17$ speed-up.

Keywords: GPGPU, RNA, secondary structure, minimum free energy.

1 Introduction

The computation of secondary structural folding of RNA or single-stranded DNA is a key element in many bioinformatics studies and, as such, has been extensively studied for many years. The firsts to propose an algorithm to predict the folding structure of RNA or DNA sequences were Waterman, Smith and Nussinov et al. [1,2]. This algorithm was based on dynamic programming with a complexity of $\mathcal{O}(n^3)$, yet their approach had several issues.

Following this pioneer work, several improvements have been done leading to different kinds of dynamic programming algorithms. We can cite: (1) the computation of the most stable structure through energy minimization running in $\mathcal{O}(n^3)$, introduced by Zuker and Stiegler [3] which outputs a single optimal structure and its corresponding energy ; (2) the computation of a partition function over all possible structures for deriving additional properties of the thermodynamic ensemble such as the base pairing probabilities of any base pair, introduced by McCaskill [4] ; (3) the computation of suboptimal structures [5] which generates all structures within a given energy range of the optimal one. Implementations of those algorithms are found in two major packages, ViennaRNA and Unafold [6,7].

Despite many huge efforts to reduce the algorithmic computational complexity, execution times are steadily increasing due to the fast growing of genomic databases and, for the last years, the relative stagnation of the microprocessor frequencies. One solution is to use multi-core systems or clusters, which can yield good performance but at a high cost. Another approach is the use of computer graphics hardware, which possibly exhibits a higher performance/cost ratio than clusters.

Indeed, the raw power of graphics processing unit (GPU) has a faster increase rate than traditional microprocessors. Moreover, recent improvements in the programmability of GPUs have opened the way to new applications from which GPUs were not initially designed for. General purpose computation on GPU (GPGPU) is now a field of research investigated in many domains requiring high performances. Among many others, successful applications include bioinformatics with the Smith-Waterman sequence alignment [8,9].

In this paper, we investigate how GPUs can be used to accelerate the computation of the minimum free energy of RNA or DNA sequence folding. We use the implementation of the Unafold package given in the function *hybrid-ss-min* [7]. This function is intensively used in different programs of the Unafold package and represents the most time consuming part. We show that adding a graphical board can speed-up the whole program by a factor $\times 17$ compared to a sequential execution on a one-core microprocessor.

Although the RNA folding algorithm studied uses dynamic programming just like the Smith Waterman algorithm, they should not be confused. Both algorithms are very different, thus previous GPU implementations of the Smith Waterman algorithm [8,9] did not prefigure the feasibility of an efficient GPU implementation here. On the contrary, its complexity in terms of memory access patterns and parallelization issues makes it a real challenge.

The rest of paper is organized as follows: In Section 2, we introduce the folding algorithm. In section 3, the GPU implementation of the folding algorithm is explained. Finally, section 4 gives the performance results obtained on different platforms.

2 Folding Algorithm

This section briefly exposes the principles of the folding algorithm as implemented in the Unafold package in the function *hybrid-ss-min* [7].

2.1 RNA Structure

RNA or Ribonucleic acid is a chain of nucleotide units. There are four different nucleotides, also called *bases*: adenine (A), cytosine (C), guanine (G) and uracil (U). Two nucleotides can form a bond thus forming a *base pair*, according to the Watson-Crick complementarity: A with U, G with C; but also the less stable combination G with U, called wobble base-pair. All the base pairs of a sequence force the nucleotide chain to fold into a system of different recognizable domains like hairpin loops, bulges, interior loops or stacked regions. This is called the *secondary structure* of the sequence. The different loop types are introduced in Fig. 1. The secondary structure can also form complex patterns like *pseudoknots* which consist of two base pairs $i \cdot j$ and $k \cdot l$ that do not verify the nesting property $i < j < k < l$. The secondary structure is often determinant in the functional role of the RNA molecule.

2.2 Energy Model

The algorithm is designed to find the most stable structure of a RNA sequence. It is used in many bioinformatics pipelines such as the search of micro RNAs where the stability of the secondary structure is an important feature.

A secondary structure is described by a list of base pairs $i \cdot j$ where each base forms at most one pair. The algorithm is based on a decomposition of the secondary structure into its constituent loops. Each loop is associated with an experimentally measured energy according to its sequence, length and type. The stability (free energy) of a structure is the sum of the energies of all its loops.

In the dot bracket representation given in Fig. 1, an unpaired base is depicted by a dot, and a pair by a matching pair of parenthesis. In the model used, matching pairs of parenthesis have to be well nested, i.e there are no pseudoknots. This restriction is a requirement to allow a relatively fast dynamic programming approach as the one developed by Zuker and Stiegler. Indeed, it ensures that the secondary structure of each subsequence i, j can be computed independently from the rest of the sequence, a required feature for dynamic programming.

2.3 Algorithm

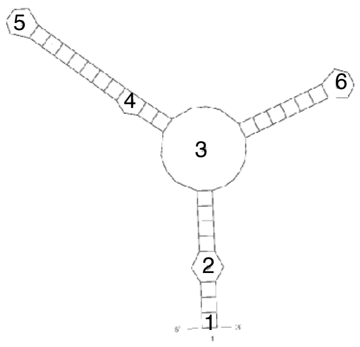
The dynamic programming algorithm uses three tables: $Q'_{i,j}$ is the minimum energy of folding of a subsequence i, j given that bases i and j form a base pair; $Q_{i,j}$ and $QM_{i,j}$ are the minimum energy of folding of the subsequence i, j assuming that this subsequence is inside a multiloop and that it contains respectively at least one and two base pairs. A simplified model of the recursion relations can be written as:

$$Q'_{i,j} = \begin{cases} \min \begin{Bmatrix} Eh(i,j) \\ Es(i,j) + Q'_{i+1,j-1} \\ \min\limits_{k,l \in]i;j[^2} Ei(i,j,k,l) + Q'_{k,l} \\ QM_{i+1,j-1} \end{Bmatrix} & \textit{if pair } i \cdot j \textit{ is allowed} \\ \infty & \textit{if pair } i \cdot j \textit{ is not allowed} \end{cases} \tag{1}$$

$$QM_{i,j} = \min_{i<k<j} (Q_{i,k} + Q_{k+1,j}) \tag{2}$$

$$Q_{i,j} = \min \left\{ QM_{i,j}, \min(Q_{i+1,j}, Q_{i,j-1}), Q'_{i,j} \right\} \tag{3}$$

$Eh(i,j)$ $Ei(i,j,k,l)$ and $Es(i,j)$ are respectively the energies of:



```
AAAAAAGGGAAAAGAACAAAGGAGACUCUUCUCCUUUUUCAAAGGAAGAGGAGACUCUUUCAAAAAUCCCUCUUUU
((((.(((((...(((.((((((((....)))))))))))...(((((((....))))))).....))))).)))) (-24.5)
```

Fig. 1. Secondary structure. The secondary structure begins in 1 with stacked base pairs (two closing base pairs with both sides of the loop of length zero). 2 is an interior loop (two closing base pairs with both sides non null). 3 shows a multiloop (several closing base pairs). 4 is a bulge loop (two closing base pairs with one loop side of length zero and the other greater than zero. 5 and 6 are hairpin loops (one closing base pair). The structure can also be written in a dot bracket representation where an unpaired base is a dot and a base pair is a matching pair of parenthesis. The free energy of the structure (-24.5) is the sum of the energies of its constituent loops.

– $Eh(i,j)$: a hairpin loop closed by the pair $i \cdot j$.
– $Ei(i,j,k,l)$: an interior loop formed by the two base pairs $i \cdot j$, $k \cdot l$.
– $Es(i,j)$: two stacked base pairs $i \cdot j$ and $(i+1) \cdot (j-1)$.

These functions compute energies through the use of lookup tables containing energy parameters according to the size and sequence of the loop.

E_j being the minimum free energy of subsequence $1 \ldots j$, the minimum free energy E_n of the whole sequence is then obtained through the recursion:

$$E_j = \min \left\{ E_{j-1}, \min_{1<k<j} (E_{k-1} + Q'_{k,j}) \right\} \tag{4}$$

Dynamic programming using this recursion computes the minimum free energy of a sequence of length n in $\mathcal{O}(n^2 \cdot L^2 + n^3)$ by restricting the loop size of interior loops to L. The corresponding secondary structure is then obtained by a trace-back procedure.

3 GPU Implementation

3.1 Architecture and Programming

GPUs are massively parallel architectures providing cheap high performance computing. We choose in our work CUDA as it combines high performance with the ease of use of a C-like environment [10]. The latest NVIDIA GPU, the GT200, is divided into 30 multiprocessors each being a SIMD unit of 8 32-bit processors. A GPU procedure is a *kernel* called on a set of threads, divided in a *grid* of *blocks* each running on a single multiprocessor. Furthermore *Blocks* are divided in warps of 32 threads that must execute the same instruction simultaneously. Thus, *branching* (if-then-else control flow instructions) does not impact performance as long as each thread within a warp take the same code path. Moreover, only threads within a block can be synchronized and can share the fast on-chip shared memory. One key difference with a traditional CPU implementation is that the programmer has to explicitly handle several memory spaces of different performance, size, scope and lifetime: global, texture, constant and shared memory as well as registers.

3.2 Parallelization Scheme

Algorithm 1 shows the main loops of the computation along with the several ways to expose parallelism. We chose a mixed approach: we compute the minimum free energy of folding of several sequences in parallel, each one being itself parallelized. According to this parallel scheme, we can provide the GPU with many independent tasks together with a low memory consumption. The number of sequences being computed simultaneously is adapted according to their length: one large sequence can provide enough independent tasks to the GPU whereas small ones have to be computed by groups. We also implemented a multi-GPU algorithm by dividing work among GPUs at the coarse-grained level, each GPU computes a different group of sequences.

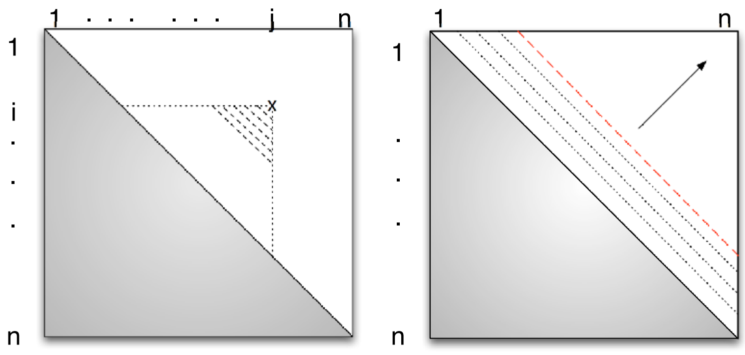


Fig. 2. Left: Data dependency relationship. Each cell of the matrix contains the three values Q', QM and Q. As subsequence i, j is the same as subsequence j, i only the upper half of the matrix is needed . The computation of cell i, j needs the lower left dashed triangle and the two vertical and horizontal dotted lines. **Right: Parallelization.** According to the data dependencies, all cells along a diagonal can be computed in parallel from all previous diagonals.

Figure 2 shows the data dependencies coming from the recursion (1) to (3). They imply that, given all previous diagonals, all cells of a diagonal can be processed independently. Three kernels are designed for the computation of $Q'_{i,j}$, $QM_{i,j}$ and $Q_{i,j}$, according to equations (1) to (3). Each one computes one diagonal of several sequences. The whole matrix is then processed sequentially through a loop over all diagonals. The next step corresponding to equation (4) is a combination of reductions (search of the minimum of an array) which is parallelized in another kernel. The final step, the traceback procedure for computing the secondary structure, is currently left on the CPU as its execution time is far lower.

3.3 Optimization Key Points

Memory accesses are the bottleneck of the implementation. Here, the data are divided into three groups: the base sequence, the three tables Q', QM,Q and the energy parameters needed for the computation of loop energies. Maximum performances are obtained when available memory resources are used to their maximum and when the best match between the different memory spaces and type of data are found. Here, the texture memory is used for the sequence and parts of the tables which both show some spatial locality in their access pattern, as for the computation of one cell $QM_{i,j}$ where equation (2) shows that accesses to all elements of a line and column of matrix Q have to be made. For energy parameters, the best choice is the constant memory. However, its small size compels us to also employ the global memory for the least used ones. Lastly, the shared memory is kept for storage of intermediate results in the computation.

Another important issue of the implementation comes from equation (1) which shows that the computation of table Q' is not the same for all cells: if the pair $i \cdot j$ is forbidden then cell $Q'_{i,j}$ is set to ∞. This hurts the SIMD model of GPU which, as stated section 3.1, says that in order to get full performance all threads of a warp must execute the same instruction path. To solve this issue our implementation computes on CPU an index of all the cell positions that have their base pairs allowed, which is then handed to the GPU. This increases the amount of data transferred between the CPU and the GPU but decreases *branching* in GPU kernels. Moreover CPU computation can be overlapped with GPU computation thus allowing us to better use all available resources.

We found that for maximum efficiency the parallelization has to be done up to the the finest grain achievable, to ensure the GPU reaches its maximum potential while using the less memory possible. Different levels of parallelization are exploited: parallelization across several sequences, across several cells of a diagonal, and across tasks required for the computation of a single cell itself: the search of a minimum is parallelized on several threads of a same block sharing intermediate results through shared memory.

Algorithm 1. Main function and parallelizable loops

```
1: Input: $N$ sequences of length $L$
2: Output: minimal energy of the $N$ sequences
3: Coarse-grained level: parallelization over multiple sequences
4: for sequence $s$ in $[1;N]$ do
5:    for diagonal $d$ in $[1;L]$ do
6:       Medium-grained level: parallelization over multiple cells of a diagonal
7:       for i in $[1;L-d]$ do
8:          $j \leftarrow i+d$
9:          Fine-grained level: parallelization over the minimization computation
10:         compute $Q'(i,j,s)$, $QM(i,j,s)$, $Q(i,j,s)$
11:      end for
12:   end for
13:   compute $E_L(s)$
14: end for
```

4 Results

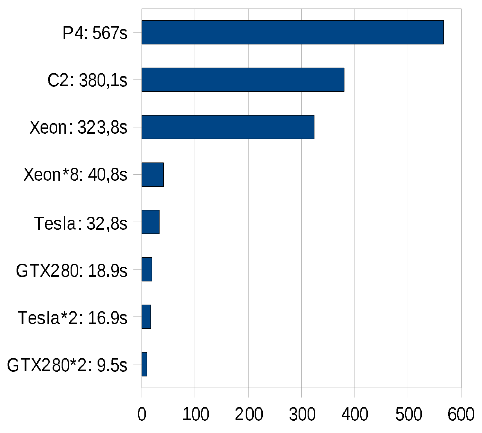
GPU and CPU implementations are both compared on different graphic cards and processors. The main testing platform is an octo-core Xeon E5430 2.66Ghz ($4 \times$ 6MB L2 cache) with 8GB RAM and two NVIDIA Tesla C870 cards, each having 16 multiprocessors. We also test older processors, a Pentium 4 3Ghz (1MB L2 cache), a Core2 6700 2.66GHz (4MB L2 cache), and the latest high-end graphic card the NVIDIA GTX280 with 30 multiprocessors.

4.1 Analysis on 120 Bases-Long Sequences

Problem specifications. A typical use of the algorithm is the computation of the secondary structures of many small RNA sequences. The search of micro RNAs in a whole genome requires, for example, to know the secondary structure of millions of sequences of length approximately 120 [11]. Therefore we first choose to test the algorithm on sequences of this length, here with 40000 randomly generated sequences.

Figure 3 reports running times in seconds and the corresponding speedup achieved by different combination of cards versus one or eight CPU cores. Our CPU multi-core implementation is done on a coarse-grained level by parallelizing the work over multiple sequences, corresponding to line 4 of algorithm 1.

Results. We achieve a speed-up of about $\times 10$ for one Tesla card versus one core of a Xeon. An interesting point is that although the algorithm was originally developed with the Tesla, it scales well with the latest graphic card. The GTX280 is 70% faster than the Tesla with a speed-up of $\times 17$ versus one core of a Xeon, which roughly corresponds to the increase of memory bandwidth between the two cards. With the two Tesla, the speed-up becomes $\times 19$, and two GTX280 get $\times 33.1$, which shows that the processing power of cards adds up well when used together.



	Xeon	Xeon x8
Tesla C870	9.9	1.2
GTX 280	17.1	2.1
Tesla C870 x2	19.2	2.4
GTX 280 x2	33.1	4.2

Fig. 3. Left: Execution Time. Time spent in seconds for the computation of the minimum free energy of 40000 randomly generated sequences of length 120, energy only (option -E of the *hybrid-ss-min* function). Processors are: P4 a pentium4 3.0Ghz (1MB cache), C2 is one core of a core2 2.66 Ghz (4MB cache), Xeon and Xeon*8 are respectively one and eight cores of Xeon 2.66Ghz (6MB cache). Graphic cards are NVIDIA Tesla C870, GTX280, bi-Tesla C870, and bi-GTX280. **Right: Corresponding speed-up.** Acceleration ratio of graphic cards versus Xeon processor, one core or octo-core configuration.

Accuracy. Our GPU implementation uses exactly the same algorithms and thermodynamic rules as Unafold, thus the results and accuracy obtained on GPU is exactly the same as the standard CPU Unafold function.

Performance / cost analysis. When using a parallelized version of the algorithm on the eight CPU cores, speed-ups are much less (Figure 3), yet the performance/cost ratio is clearly in favor of the GPU implementation. Indeed our results show that a system with two GTX280, easily doable for 2500 euros, would be roughly equivalent to four octo-core computers costing a total of more than 8000 euros.

As for standard computers at everyone disposal, the advantage of GPUs is also obvious: considering every systems are now dual-cores, adding a GTX280 would allow to get at least $\times 8$ performance even if both CPU cores are used, at a cost of about 400 euros.

4.2 Analysis across Varying Sequence Lengths

The algorithm is then experimented upon with various sequence lengths. Speed-up of Tesla and GTX280 versus one Xeon processor core are showed in figure 4. It should first be noted that the GTX 280 is always at least 50% faster than the Tesla except for very long sequences, where it begins to lack memory (Tesla has 1.5 GB whereas GTX 280 has 1.0 GB). We see that performance is good for short

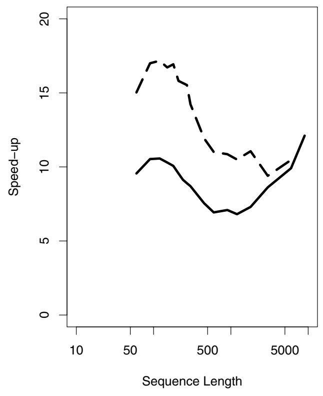


Fig. 4. Speed up comparison. Speed up of Tesla C870 and GTX 280 graphic card versus one core of a 2.66 Ghz Xeon for randomly generated sequences of different lengths. Solid line is Tesla C870, dashed line is GTX 280.

sequences (Tesla gets $\times 10$ speed-up), then it comes to a minimum for 1000 bases long sequences (Tesla gets $\times 7$) and it rises again for very long sequences ($\times 12$ for Tesla with sequence of length 9000). This comes from the fact that different portions of the code do not have the same computational complexity and GPU efficiency. With n the length of a sequence, QM computation is in $\mathcal{O}(n^3)$ whereas Q' computation is in $\mathcal{O}(n^2)$. The efficiency of the $\mathcal{O}(n^2)$ part decreases when n increases due to different memory access patterns, which explains the decrease in performance. The $\mathcal{O}(n^3)$ part of the algorithm is always very efficient on GPU but only becomes a preponderant part of the algorithm for long sequences, which explains the overall speed up increase we observe for long sequences.

4.3 Comparison Against GTfold

A.Mathuriya *et al.* implemented a CPU multicore algorithm for RNA secondary structure prediction which uses what we call in algorithm 1 the medium-grained level [12]. They compute in their study the folding of the HIV-1 sequence and a set of 11 Picornaviral sequences on a 32-core IBM P5-570 server. Table 1 compares the running time they obtain against our GPU implementation on one Tesla C870 card. It shows that an expensive 32-core server only gets $\times 1.6$ the performance of a single GPU.

Table 1. Running times on HIV-1 sequence (9781 nucleotides) and a set of 11 Picornaviral sequences (7124 to 8214 nucleotides), cf [12] for sequence accession numbers

	GTfold 32-core IBM P5-570	GPU Tesla C870	Unafold 1 core Xeon
HIV-1	84 s	133 s	1876 s
11 Picornavirus	480 s	765 s	7902 s

5 Future Work

This work is the first step in parallelizing RNA folding algorithm on GPU. It shows that GPUs can deliver significant speed-ups even on algorithms with complex memory access patterns.

However, although GPUs recently became easier to use, an efficient GPU implementation remains a lengthy process. For years programmers have developed purely sequential algorithms, yet it appears that future systems will become more and more highly parallel architectures. Thus, a future challenge will be to find a way to facilitate implementation of algorithms for a parallel execution; on multi-core chips using the MIMD paradigm, on GPUs using the SIMD paradigm, and the trickiest task, on a combination of both.

References

1. Waterman, M., Smith, T.: RNA secondary structure: a complete mathematical analysis. Math. Biosci. 42, 257–266 (1978)
2. Nussinov, R., Pieczenik, G., Griggs, J., Kleitman, D.: Algorithms for loop matchings. SIAM J. Appl. Math. 35(1), 68–82 (1978)
3. Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res. 9(1), 133–148 (1981)
4. McCaskill, J.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers 29(6-7), 1105–1119 (1990)
5. Wuchty, S., Fontana, W., Hofacker, I.L., Schuster, P.: Complete suboptimal folding of RNA and the stability of secondary structures. Biopolymers 49, 145–165 (1999)
6. Hofacker, I.L., Fontana, W., Stadler, P.F., Bonhoeffer, L.S., Tacker, M., Schuster, P.: Fast folding and comparison of RNA secondary structures. Monatsh. Chem. 125, 167–188 (1994)
7. Markham, N., Zuker, M.: DINAMelt web server for nucleic acid melting prediction. Nucleic Acids Research 33, W577–W581 (2005)
8. Liu, W., Schmidt, B., Voss, G., Schroeder, A., Muller-Wittig, W.: Bio-sequence database scanning on gpu. In: Proceeding of the 20th IEEE International Parallel & Distributed Processing Symposium: 2006(IPDSP 2006) (HICOMB Workshop) (2006)
9. Svetlin, M., Giorgio, V.: CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics 9 (2008)
10. Nvidia cuda, `http://developer.NVidia.com/object/cuda.html`
11. Stark, A., Kheradpour, P., Parts, L., Brennecke, J., Hodges, E., Hannon, G.J., Kellis, M.: Systematic discovery and characterization of fly microRNAs using 12 Drosophila genomes. Genome Research 17(12), 1865 (2007)
12. Mathuriya, A., Bader, D., Heitsch, C., Harvey, S.: GTfold: A Scalable Multicore Code for RNA Secondary Structure Prediction. Technical report, Georgia Institute of Technology (2008)

Parallel Calculating of the Goal Function in Metaheuristics Using GPU

Wojciech Bożejko, Czesław Smutnicki, and Mariusz Uchroński

Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
{wojciech.bozejko,czeslaw.smutnicki,mariusz.uchronski}@pwr.wroc.pl

Abstract. We consider a metaheuristic optimization algorithm which uses single process (thread) to guide the search through the solution space. Thread performs in the cyclic way (iteratively) two main tasks: the goal function evaluation for a single solution or a set of solutions and management (solution filtering and selection, collection of history, updating). The latter task takes statistically 1-3% total iteration time, therefore we skip its acceleration as useless. The former task can be accelerated in parallel environments in various manners. We propose certain parallel small-grain calculation model providing the *cost optimal* method. Then, we carry out an experiment using Graphics Processing Unit (GPU) to confirm our theoretical results.

1 Introduction

Almost all combinatorial optimization tasks formulated for scheduling problems are strongly NP-hard. Currently known exact solution algorithms (dedicated to find global optimum) own exponential computational complexity which causes unacceptable long solution time for instances that come from practice. In this context one can propose two, not mutually conflicted, approaches, which allow one to solve large-size instances in acceptable time: (1) approximate methods (chiefly metaheuristics), (2) parallel methods. The best hybrid combination of both is the one which we really needed.

The most promising metaheuristic algorithms search solution space in a certain intelligent way. Quality of the best solutions generated by these algorithms strongly depends on the number of analyzed solution, and thus on the running time. Time and quality have opposing tendency in such a sense, that finding a better solution requires a significant computation time growth. Through the parallel processing one can increase the number of checked solutions (per time unit). In this paper there are proposed several solutions algorithms dedicated to a single solution analysis employed in widely used metaheuristics.

In the scope of a single-thread search, dedicated fundamentally for uniform multiprocessor system of small granularity, one can distinguish a few parallel approaches taking into account various design technologies and different needs applied by modern discrete optimization algorithms, namely: (a) single solution analysis (dedicated for simulated annealing SA, simulated jumping SJ, random search RS), (b) local neighborhood analysis (for tabu search TS, adaptive memory search AMS, descending search DS), (c) analysis of population of distributed solutions (for genetic approach GA, scatter search SS). In each case special attention should be paid to efficiency, cost and speedup of methods depending on the used parallel computing environment. For each algorithm, theoretical evaluation of its numerical properties as well as comparative analysis of potential benefits from proposed approaches are expected. In this paper we deal chiefly with the approach (a).

In this paper we use the following notions which are fundamental in the parallel computing area, see e.g. [4]: theoretical parallel architectures, theoretical models of parallel computations, granularity, threads, cooperation, speed up, efficiency, cost, cost optimality, computational complexity, real parallel architectures and parallel programming languages.

This work constitutes the continuation of authors research on constructing efficient algorithms applied to solve hard combinatorial problems ([2,5,6]).

2 Permutation Flow Shop Problem

We consider, as the test case, the well-known in the scheduling theory, strongly NP-hard problem, called the permutation flow-shop problem with the makespan criterion and denoted by $F||C_{max}$. Skipping consciously the long list of papers dealing with this subject we only refer the reader to the recent reviews and the best up-to-now algorithms [5,6].

The problem has been introduced as follows. There is n jobs from a set $J = \{1, 2, \ldots, n\}$ to be processed in a production system having m machines, indexed by $1, 2, \ldots, m$, organized in the line (sequential structure). A single job reflects one final product (or sub product) manufacturing. Each job is performed in m subsequent stages, in a common way for all tasks. The stage i is performed by machine i, $i = 1, \ldots, m$. Each job $j \in J$ is split into a sequence of m operations $O_{1j}, O_{2j}, \ldots, O_{mj}$ performed on machines in turn. The operation O_{ij} reflects processing of job j on the machine i with the processing time $p_{ij} > 0$. Once started job cannot be interrupted. Each machine can execute at most one job at a time; each job can be processed on at most one machine at a time.

The sequence of loading jobs into system is represented by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ on the set J. The optimization problem is to find the optimal sequence π^* so that

$$C_{max}(\pi^*) = \min_{\pi \in \Pi} C_{max}(\pi). \quad (1)$$

where $C_{max}(\pi)$ is the makespan for permutation π and Π is the set of all permutations. Denoting by C_{ij} the completion time of job j on the machine i we have $C_{max}(\pi) = C_{m,\pi(n)}$. Values C_{ij} can be found by using the recursive formula

$$C_{i\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i\pi(j)}, \quad i = 1, 2, \ldots, m, \quad j = 1, \ldots, n, \quad (2)$$

with initial conditions $C_{i\pi(0)} = 0$, $i = 1, 2, \ldots, m$, $C_{0\pi(j)} = 0$, $j = 1, 2, \ldots, n$. Computational complexity of (2) is $O(mn)$.

Values C_{ij} from the equation (2) can be also determined by using a graph model of the flow shop problem. For a given sequence of jobs execution $\pi \in \Pi$ we create a graph $G(\pi) = (M \times N,\ F^0 \cup F^*)$, where $M = \{1, 2, \ldots, m\}$, $N = \{1, 2, \ldots, n\}$, $F^0 = \bigcup_{s=1}^{m-1} \bigcup_{t=1}^{n} \{((s, t), (s + 1, t))\}$ is a set of technological arcs (vertical) and $F^* = \bigcup_{s=1}^{m} \bigcup_{t=1}^{n-1} \{((s, t), (s, t + 1))\}$ is a set of sequencing arcs (horizontal). Arcs of the graph $G(\pi)$ have no weights, but each vertex (s,t) has weight $p_{s,\pi(t)}$. The time C_{ij} of completing job $\pi(j)$, $j = 1,2,...,n$ on the machine i, $i = 1,2,...,m$ equals the length of the longest path from the vertex (1,1) to the vertex (i,j), including the weight of the last one. For the $F||C_{max}$ problem the value of the criterion function for the fixed sequence π equals the length of the critical path in the graph $G(\pi)$.

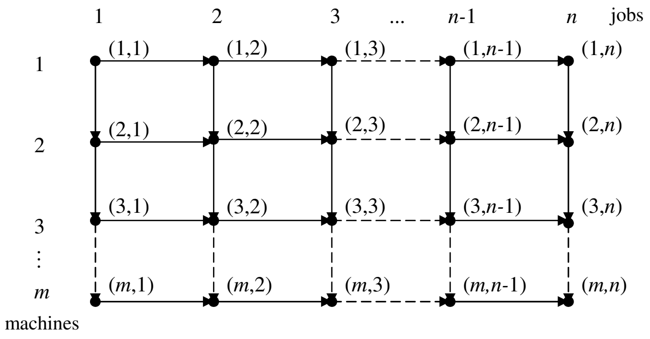


Fig. 1. Graph $G(\pi)$

3 Searching

We consider a solution method which uses only one thread to manage the search process. The process executes cyclic iterations consisting of: (1) numerical calculations (i.e. the goal function value determination), (2) managing functions (i.e. solution selection, calculations memory realization, solution acceptation). Activities connected with managing take statistically 1-3% of the iteration's time. Therefore, its acceleration by using parallel environment gives us insignificant benefit. From the other side, the numerical calculations acceleration by implementing in the parallel or distributed architecture may significantly improve the efficiency of the solution space's search algorithm.

We take advantage of the following well-known facts for the PRAM parallel computer model:

Fact 1. *Sequence of prefix sums $(y_1, y_2, ..., y_n)$ of input sequence $(x_1, x_2, ..., x_n)$ such, that*

$$y_k = y_{k-1} + x_k = x_1 + x_2 + ... + x_k \text{ for } k = 2, 3, ..., n$$

where $y_1 = x_1$ can be calculated in time $O(\log n)$ on the EREW PRAM machine with $O(n/\log n)$ processors.

From what we have stated above we can assume that the sum of n values can be calculated in time $O(\log n)$ on $O(n/\log n)$ – processors EREW PRAM machine.

Fact 2. *The minimal and the maximal value of input sequence $(x_1, x_2, ..., x_n)$ can be determined in the time $O(\log n)$ on the EREW PRAM machine with $O(n/\log n)$ processors.*

Fact 3. *The value of $y = (y_1, y_2, \ldots, y_n)$ where $y_i = f(x_i)$, $x = (x_1, x_2, \ldots, x_n)$ can be calculated on the CREW PRAM machine with n processors in a time $O(c) = O(1)$, where c is a time needed to calculate the single value of $y_i = f(x_i)$.*

Fact 4. *The problem formulated in the previous fact can be calculated in the time $O(\log n)$ on $O(n \log n)$ processors.*

If we do not have such a big number of processors we can use such a fact to keep the same cost:

Fact 5. *If the algorithm A works on p – processors PRAM in the time t, then for every $p' < p$ exists an algorithm A' for the same problem which works on p' – processors PRAM in time $O(pt/p')$.*

In each iteration we have to find a goal function value for a *single* fixed π. Calculations can be spread into parallel processors in a few ways.

Theorem 1. *For a fixed π the value of criterion function for problems $F||C_{max}$ and $F||C_{sum}$ can be found on the CREW PRAM machine in the time $O(n+m)$ by using m processors.*

Proof. Without the loss of generality one can assume that $\pi = (1, 2, \ldots, n)$. Calculations of $C_{i,j}$ by using (2) have been clustered. Cluster k contains values C_{ij} such that $i + j - 1 = k$, $k = 1, 2, \ldots, n + m - 1$ and requires at most m processors. Clusters are processed in the order $k = 1, 2, \ldots, n+m-1$. The cluster k is processed in parallel on at most m processors. The calculations sequence is shown in Fig. 2 on the background of the grid graph commonly used for the flow shop problem. Values linked by dashed lines constitute a single cluster. The value of C_{max} criterion is simple $C_{m,n}$. To calculate $C_{sum} = \sum_{j=1}^{n} C_{m,j}$ we need to add n values $C_{m,j}$, which can be done sequentially in n iterations or in parallel by using m processors with the complexity $O(n/m + \log m)$. Finally, the computational complexity of determining the criterion value for $F||C_{max}$ and $F||C_{sum}$ problems is $O(n + m)$ by using m processors.

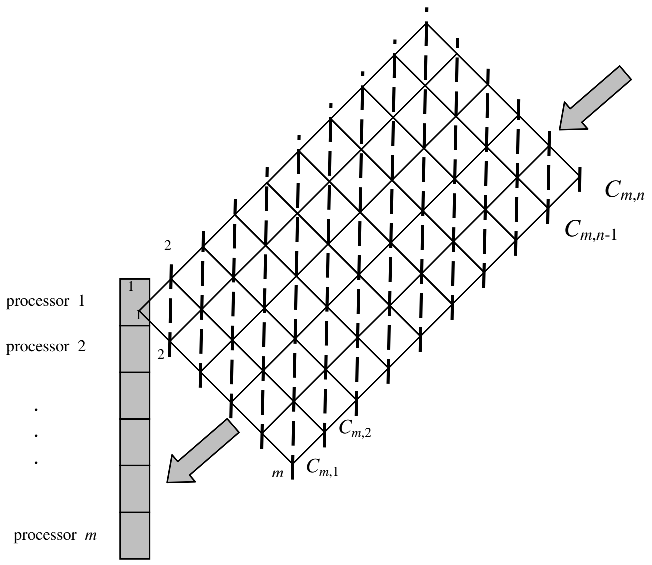


Fig. 2. Computing C_{ij} order using m threads

Fact 1. Speedup of method from Theorem 1 is $O(\frac{nm}{n+m})$, efficiency is $O(\frac{n}{n+m})$[1].

Theorem 2. *For a fixed π the value of criterion function for problems $F||C_{max}$ and $F||C_{sum}$ can be found on the CREW PRAM machine in the time $O(n+m)$ by using $O(\frac{nm}{n+m})$ processors.*

Proof. Without the loss of generality one can assume that $\pi = (1, 2, \ldots, n)$. We based on the scheme of calculations shown in Fig. 2. Let $p \leq m$ be the number of used processors. The calculation process will be carried out for *levels* $k = 1, 2, \ldots, d$, $d = n+m-1$ in this order. On the level k we perform a calculation of n_k values $C_{i,j}$ such that $i + j - 1 = k$, $\sum_{k=1}^{d} n_k = nm$.

We cluster n_k elements on the level k into $\left\lceil \frac{n_k}{p} \right\rceil$ groups; first $\left\lfloor \frac{n_k}{p} \right\rfloor$ groups contain p elements each, whereas the remaining elements (at most p) belong to the last group. Parallel computations on the level k are performed in the time $O(\left\lceil \frac{n_k}{p} \right\rceil)$. The total calculation time is equal to the sum over all levels and is of order

$$\sum_{k=1}^{d} \left\lceil \frac{n_k}{p} \right\rceil \leq \sum_{k=1}^{d} \left(\frac{n_k}{p} + 1 \right) = \frac{nm}{p} + d = \frac{nm}{p} + n + m - 1. \qquad (3)$$

We are seeking for the number of processors p, $1 \leq p \leq m$, for which efficiency of parallel algorithm is $O(1)$, which ensures cost optimality of the method. The

[1] Evaluation is true with a certain constant multiplier.

value p can be found from the following condition

$$\frac{1}{p}\,\frac{nm}{\frac{nm}{p}+n+m-1}=c=O(1) \tag{4}$$

for some constant $c<1$. After a few simple transformations of (4) we get

$$p=\frac{nm}{n+m-1}\left(\frac{1}{c}-1\right)=O(\frac{nm}{n+m}). \tag{5}$$

Setting $p=O(\frac{nm}{n+m})$ we obtain the total calculation time of C_{ij} values equals

$$O(\frac{nm}{p}+n+m-1)=O(\frac{nm}{\frac{nm}{n+m}}+n+m)=O(n+m). \tag{6}$$

Fact 2. Speedup of the method based on Theorem 2 is $O(\frac{nm}{n+m})$, cost is $O(nm)$.

The method is cost optimal and allows one to control efficiency as well as speed of calculations by choosing the number of processors and adjusting the parameters of calculations to the real number of parallel processors existing in the system. Besides, Theorem 2 provides the "optimal" number of processors that ensures the cost optimality of this method. This number can be set by a flexible adaptation of the number of processors to both sizes of the problem, namely n and m simultaneously.

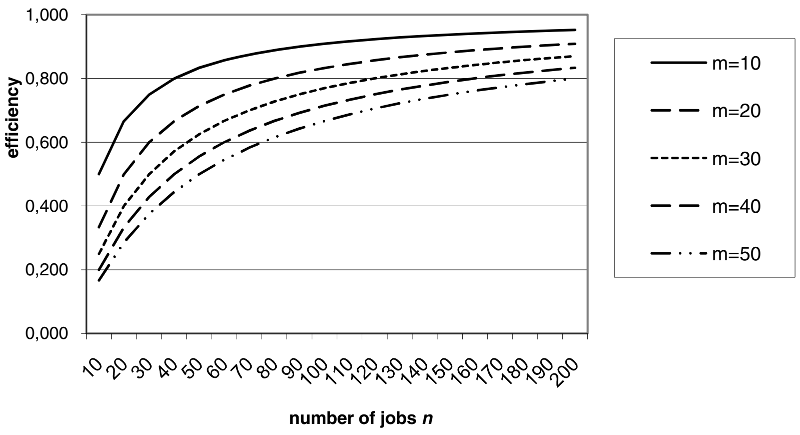


Fig. 3. Efficiency of the method from Theorem 2

4 Experimental Results

The parallel algorithm for the considered problem of calculating makespan in permutation flow shop problem was coded in C (CUDA) for GPU, ran on the

Tesla C870 GPU (512 GFLOPS) with 128 streaming processor cores and tested on the benchmark problems of Taillard. The benchmark set contains 120 particularly hard instances of 12 different sizes. For each size (group) $n \times m$: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, 500×20, a sample of 10 was provided. The considered algorithm showed as Algorithm 1 uses m GPU processors for calculating makespan. Its sequential version is obtained by assigning $p = 1$ and it is also executed on GPU. Algorithm 2 presents details of the parallel method coded in CUDA.

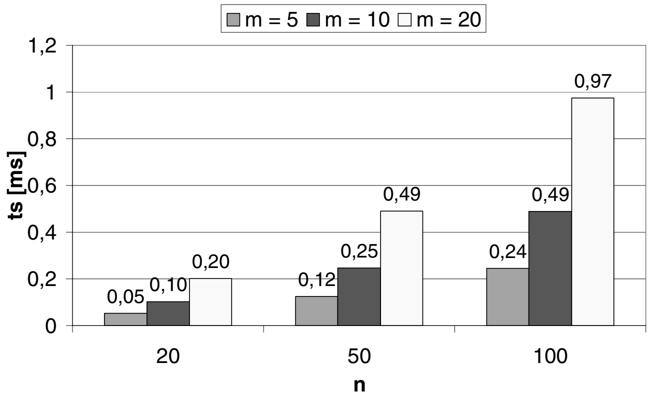


Fig. 4. The one thread algorithm computational times

Algorithm 1. Parallel algorithm of the makespan calculating

```
**parfor** $i = 1$ to $p$ (for each processor)
  **for** $j = 1$ to $n + m - 1$ **do**
    $x = j - i + 1$
    **if** $x \geq 1$ and $x \leq n$ **then**
      $C_{x,i} = \max\{C_{x-1,i}, C_{x,i-1}\} + p_{x,i}$
    **end if**
  **end for**
**end of parfor.**
```

Each multiprocessor of the TESLA C870 GPU has on-chip memory of the four following types:

- one set of local 32-bit *registers* per processor,
- a parallel data cache or *shared memory* that is shared by all scalar processors cores and is where the shared memory space resides,
- a read-only *constant cache* that is shared by all scalar processor cores and speeds up reads from the constant memory space, which is a read-only region of device memory,
- a read-only *texture cache* that is shared by all scalar processor cores and speeds up reads from the texture memory space, which is a read-only region of device memory.

Algorithm 2. Parallel algorithm for makespan calculating coded in CUDA

```
__global__ void cmax(int *c, int n, int m, int *cmax) {
  int idy;
  int idx = blockIdx.x * blockDim.x + threadIdx.x + 1;
  if(idx<=m) {
    for(int j=1;j<m+n;j++) {
      idy = j - idx + 1;
      if(idy>=1&&idy<=n) {
        c[idx*(n+1)+idy]=max(c[(idx-1)*(n+1)+idy],
        c[idx*(n+1)+(idy-1)]) + tex2D(tex,idy ,idx);
        if(idx==m&&idy==n) cmax[0]=c[idx*(n+1)+idy];
} } } }

int main() { //Kernel invocation
  int blockSize = 16;
  int nBlocks = M/blockSize + (M%blockSize == 0?0:1);
  cmax<<< nBlocks, blockSize >>> (devC, N, M, devCmax);
}
```

There is no need to copy table with processing times before each calculating of makespan so this table was copied once to very fast read-only texture memory. Therefore timings are measured without this preparing time. Texture memory is cached. Table C was allocated in global memory. After calculations makespan on GPU value of C_{max} is copied to the CPU. This operation is very fast (only one-element table is copied).

The sequential algorithm using one GPU processor was coded with the aim of determining the speedup value which can be obtained by a parallel algorithm. Table 1 shows computational times for the sequential and the parallel algorithm as well as speedup. The value of relative speedup s can by found by the following

Table 1. Experimental results for Taillard's instances

$n \times m$	p	t_p $[ms]$	t_s $[ms]$	speedup s
20×5	5	0.0271	0.0526	1.94
20×10	10	0.0309	0.1022	3.31
20×20	20	0.0386	0.2014	5.22
50×5	5	0.0480	0.1244	2.59
50×10	10	0.0518	0.2469	4.76
50×20	20	0.0601	0.4909	8.16
100×5	5	0.0835	0.2449	2.93
100×10	10	0.0874	0.4885	5.59
100×20	20	0.0968	0.9740	10.06
200×10	10	0.1582	0.9716	6.14
200×20	20	0.1697	1.9403	11.43
500×20	20	0.3919	4.8392	12.35

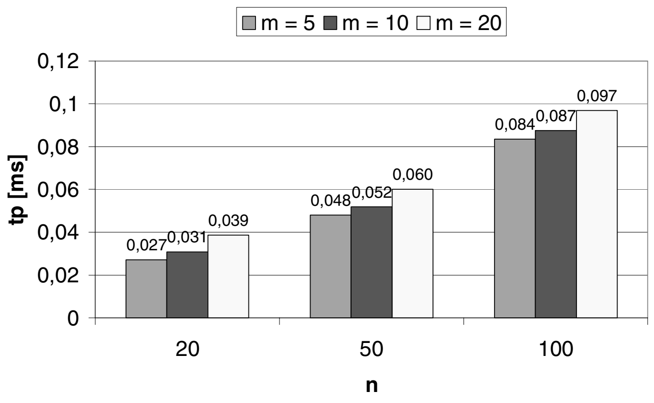


Fig. 5. The parallel algorithm computational times

expression $s = \frac{t_s}{t_p}$, where t_s constitutes the computational time of sequential algorithm and t_p - computational time of parallel algorithm. Figure 4 shows computational times of the sequential algorithm for different sizes of problem. The algorithm computational time increases with the size of a problem. For the fixed number of jobs 100% increasing of the number of machines results in 100% increasing computational time.

Figure 4 shows computational times of the parallel algorithm for different sizes of the problem. For the fixed number of jobs the increase of the number of machines results in a small computational time increase. Increase size of problem results in increasing the speedup of parallel algorithm in comparison with the sequential algorithm. Figure 6 confirms theoretical effectiveness on the basis of Theorem 2 shown on Figure 3. Obtained experimental results are fully common with the theoretical analysis.

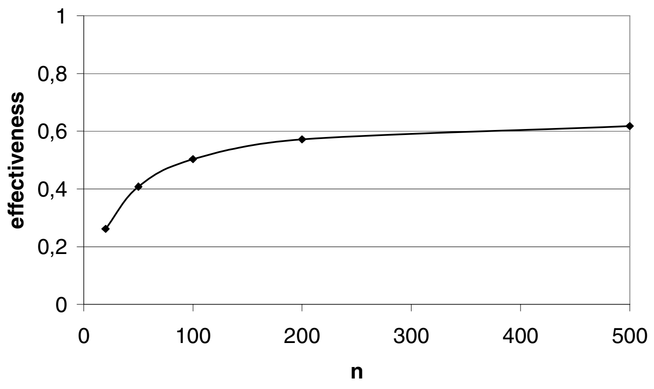


Fig. 6. Experimental effectiveness

5 Conclusions

We propose taking advantage of modern many-core computing processors GPU to accelerate local search algorithm's work. Results, for a classic flow shop scheduling problem, allow us to obtain a speedup which is proportional to the number of machines m from the problem definition. Theorems presented in this paper can be easily extended to the EREW PRAM model, with exclusive read, which requires an additional $O(\log n)$ time, however the architecture of the used GPU allows us to implement CREW algorithms (with a possibility of concurrent read). On the other side the shared memory usage is connected with a huge time latency (400-600 cycles of the clock) comparing to the local memory of a processor or the textures (constant) memory. Therefore, a further acceleration of the algorithm is possible under condition of the methods adaptation to the GPU memory access specific.

References

1. Aarts, E.H.L., Verhoeven, M.: Local search. In: Dell'Amico, M., Maffioli, F., Martello, S. (eds.) Annotated bibliographies in Combinatorial Optimization, Wiley, pp. 163–180. Wiley, Chichester (1997)
2. Bożejko, W.: Parallel scheduling algorithms, PhD Thesis, Report 29/2003, Institute of Engineering Cybernetics, Wrocław University of Technology 1–205 (2003)
3. Fiechter, C.N.: A parallel tabu search algorithm for large traveling salesman problems. Discrete Applied Mathematics 51, 243–267 (1994)
4. Grama, A., Kumar, V.: State of the Art in Parallel Search Techniques for Discrete Optimization Problems. IEEE Transactions on Knowledge and Data Engineering 11, 28–35 (1999)
5. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research 91, 160–175 (1996)
6. Nowicki, E., Smutnicki, C.: Some aspects of scatter search in the flow-shop problem. European Journal of Operational Research 169, 654–666 (2006)
7. Porto, S.C., Ribeiro, C.C.: Parallel tabu search message passing synchronous strategies for task scheduling under precedence constraints. Journal of Heuristics 1, 207–223 (1995)
8. Porto, S.C., Ribeiro, C.C.: A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. International Journal of High Speed Computing 7, 45–71 (1995)
9. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)

Author Index