PROCEEDINGS OF THE
2007 HIGH PERFORMANCE COMPUTING & SIMULATION CONFERENCE
(HPCS 2007)

In conjunction with
The 21st EUROPEAN CONFERENCE ON MODELLING AND SIMULATION
(ECMS 2007)

June 4th – 6th, 2007
Prague, Czech Republic

Edited by:

Waleed W. Smari

Co-Sponsored by SCS-Europe, IEEE Germany, ASIM, EUROSIM, CASS, JSST, PTSK, TSS and In Cooperation with the IEEE Computer Society Technical Committee on Parallel Processing (TCPP)

ORGANIZING COMMITTEE

Honorary General Chair:

Geoffrey C. Fox, Indiana University, USA

Honorary General Chair:

Domenico Talia, DEIS, Universta' della Calabria, Italy

Mads Nygård, Norwegian Univ. of Science and Technology, Norway

Program Chair:

Waleed W. Smari, University of Dayton, Ohio, USA

Tutorials Chair:

Claudia Leopold, University of Kassel, Germany

Special Sessions & Workshops Co-Chairs:

Frederick C. Harris, Jr., University of Nevada, Reno, USA

Sergiu Dascalu, University of Nevada, Reno, USA

Panels Chair:

Yudith Cardinale, Universidad Simón Bolívar, Venezuela

Awards Chair:

Ratan Guha, University of Central Florida, USA

Registration & Publications Chair:

Martina-Maria Seidel, ECMS office, Germany

Conference Web Master:

Seung-Yun Kim, University of Dayton, Ohio, USA

Conference Software Systems Manager:

Jorge Sanchez, EDAptive Computing Inc., USA

Local Arrangements:

Ivan Zelinka, Tomas Bata University in Zlin, Czech Republic

Zuzana Oplatkova, Tomas Bata University in Zlin, Czech Republic

INTERNATIONAL TECHNICAL PROGRAM COMMITTEE - HPCS 2007:

Chairperson

Waleed W. Smari, University of Dayton, Ohio, USA

ITPC Members

Hamid Abachi, Monash University, Australia
Saleh R. Al-Araji, Etisalat University College, United Arab Emirates
Marta Barría, Universidad de Valparaíso, Chile
Lars R. Bengtsson, Chalmers University of Technology, Sweden
Arndt Bode, Technical University of Munich, Germany
Laszlo Boeszoermenyi, Klagenfurt University, Austria
Per Bjarne Bro, Universidad de Talca, Chile
Edson Norberto Caceres, Universidade Federal de Mato Grosso do Sul, Brazil
Hector Cancela, Universidad de la República, Uruguay
Mario Cannataro, University of Catanzaro, Italy
Sorin Dan Cotofana, Delft University of Technology, The Netherlands
Claudia Diamantini, Universita' Politecnica delle Marche, Italy
Chyi-Ren Dow, Feng-Chia University, Taiwan
Bertil Folliot, University of Pierre and Marie Curie, Paris VI, France
Giancarlo Fortino, University of Calabria, Italy
Maria Ganzha, Elblag University of Humanities and Economy, Poland
Ratan Guha, University of Central Florida, USA
Attila Gursoy, Koc University, Turkey
Kenneth A. Hawick, Massey University - Albany, New Zealand
Gongzhu Hu, Central Michigan University, USA
Hai Jin, Huazhong University of Science and Technology, Wuhan, China
Daniel S. Katz, Louisiana State University and Jet Propulsion Laboratory, USA
Harald Kosch, University of Passau, Germany
Dieter A. Kranzlmueller, Joh. Kepler University Linz, Austria

Edmundo R. M. Madeira, University of Campinas, Brazil
Nouredine Melab, CNRS/LIFL, INRIA Futurs - Université de Lille1, France
Jean-Frederic Myoupo, University of Picardie-Jules Verne, France
Sotiris Nikoletseas, Computer Technology Institute, Patras, Greece
M. Ould-Khaoua, University of Glasgow, UK
Maria S. Perez, Universidad Politecnica de Madrid, Spain
Dana Petcu, Western University of Timisoara, Romania
Andy Pimentel, University of Amsterdam, The Netherlands
Andrew Rau-Chaplin, Dalhousie University, Canada
Thomas Rauber, University of Bayreuth, Germany
Christophe Rosenberger, ENSI-Bourges, France
Gudula Rünger, Chemnitz University of Technology, Germany
Ana Pont Sanjuán, Polytechnic University of Valencia, Spain
Erich Schikuta, University of Vienna, Austria
Stanislav G. Sedukhin, University of Aizu, Japan
Leonel Sousa, Superior Institute of Technology (IST), Portugal
Giandomenico Spezzano, ICAR-CNR, Università della Calabria, Italy
Heinz Stockinger, CERN, Swiss Institute of Bioinformatics, Switzerland
Przemyslaw Stpiczynski, Maria Curie-Sklodowska University, Poland
El-Ghazali Talbi, INRIA Futurs, Univ. des Sciences et Techn. de Lille, France
Gary Tan Soon Huat, National University of Singapore, Singapore
Petia Todorova, Fraunhofer Institute FOKUS, Germany
Pavel Tvrdik, Czech Technical University, Czech Republic
Andreas Uhl, Salzburg University, Austria
Lucian N. Vintan, Lucian Blaga University of Sibiu, Romania
Junaid A. Zubairi, SUNY at Fredonia, USA

SPECIAL SESSION ON SECURITY AND HIGH PERFORMANCE COMPUTING SYSTEMS

Organizer: **Dr. Luca Spalazzi**
Dipartimento di Ingegneria Informatica,
Gestionale e dell'Automazione
Università Politecnica delle Marche, Ancona, Italy

Session Technical Program Committee:

Massimo Benerecetti, Universita degli studi di Napoli "Federico II", Italy

Gianluca Capuzzi, Universita Politecnica delle Marche, Italy

Michael Georgeff, Monash University, Australia

Jean-Francois Lalande, LIFO, Universite d'Orleans, France

John Mylopoulos, University of Toronto, Canada

Waleed W. Smari, University of Dayton, USA

Simone Tacconi, Polizia di Stato, Italy

Carolyn Talcott, SRI International, USA

Christian Toinard, LIFO, Universite d'Orleans, France

SPECIAL SESSION ON PARALLEL AND GRID COMPUTING FOR OPTIMIZATION (PGCO 007)

Organizers: **Dr. Nouredine Melab** and **El-Ghazali Talbi**
Parallel Optimization Research Group
CNRS/LIFL, INRIA Futurs
Université de Lille1, France

Session Technical Program Committee:

E. Alba, University of Malaga, Spain

P. Bouvry, University of Luxembourg, Luxembourg

A. Lamnitchi, University of South Florida, FL, USA

J-T. Linderoth, Lehigh University, PA, USA

N. Melab, Université de Lille1, France

S. Mostaghim, Karlsruhe University, Germany

C-C. Ribeiro, University of Rio de Janeiro, Brazil

E-G. Talbi, Université de Lille1, France

SPECIAL SESSION ON PARALLEL AND DISTRIBUTED SIMULATION (PDSim 2007)

Organizer: **Dr. Karim Kabalan**
American University of Beirut, Lebanon

SPECIAL SESSION ON HIGH PERFORMANCE INFORMATION RETRIEVAL AND VISUALIZATION: ALGORITHMS AND APPLICATIONS

Organizer: **Dr. Frederick C. Harris, Jr**
University of Nevada at Reno, Nevada, USA

Session Technical Program Committee:

Brian Beck, Center for Bioinformatics, University of Nevada, Reno, USA
Tim Brown, Desert Research Institute, Nevada, USA
Kendra Cooper, University of Texas at Dallas, Texas, USA
Sergiu Dascalu, University of Nevada, Reno, Nevada, USA
Walter Dosch, University of Luebeck, Germany
Phil Goodman, University of Nevada Medical School, Nevada, USA
Vic Grout, University of Wales, Wrexham, UK
Marcel Karam, American University, Beirut, Lebanon
Andrew Rau-Chaplin, Dalhousie University, Canada
Pierre Tiako, Langston University, Oklahoma, USA
Gregory Vert, University of Nevada, Reno, Nevada, US

HPCS 2007 ADDITIONAL REVIEWERS:

Mike McMahon, University of Nevada, Reno, Nevada, USA

GREETINGS FROM THE GENERAL CO-CHAIRS OF HPCS 2007

Welcome to the Czech Republic, welcome to Prague, and welcome to HPCS 2007. This is the 5th year the conference takes place. As it has matured into an esteemed venue for publication and discussion of knowledge in the corresponding areas, we are very happy to serve as this conference's General Co-Chairs. The conference will address several issues in modelling and simulation of high performance and large scale computing systems that today play a key role in science and industry.

Very hearty thanks go to the Program Chair, Prof. Waleed W. Smari, and his colleagues for putting together such a broad, well designed and interesting program. A big thanks also goes to those who have set up all the related tutorials, special sessions, workshops and panels. We will also use the opportunity to thank the conference's local organizers from the Thomas Bata University in Zlín as well as ECMS.

Hence, enjoy the presentations and discussions; use the opportunity to meet interesting researchers, and enjoy the wonderful setting that the city of Prague gives this conference. Last but not least, welcome back also next year to HPCS 2008.

Mads Nygård
Trondheim, Norway

Domenico Talia
Rende, Calabria, Italy

April 2007

THE 2007 HPCS FOREWORD

On behalf of the organizers and International Program Committee, I would like to welcome you to the 2007 High Performance Computing and Simulation (HPCS 2007) Conference held in Prague, Czech Republic, June 4-6, 2007, in conjunction with ECMS 2007. This conference will provide a dynamic forum to address, explore, and exchange information, knowledge, and experiences in the state-of-the-art in high performance computing systems, their modelling and simulation, design and use, and impact. HPCS brings together researchers, scientists, engineers, practitioners, educators, and students from many nations and backgrounds to exchange their insights, breakthroughs, and research results about aspects of these systems and their technologies; to discuss challenges encountered in government, industry, and academe; and to seek new and innovative solutions. Additionally, we hope that the conference will present opportunities for many open technical interchanges in individual and group settings on key technology issues, during the conference and the potential for future collaborations among the participants, afterwards.

Current research in university and industry provides a new generation of HPC systems to create fully interconnected communities of interest and practice with decision quality information in compressed time cycles. Through modelling and simulation, knowledge sharing and discovery, and just-in-time global grid-based information processing, individuals and groups will work together and make better, not just faster, decisions. The technologies and research presented in HPCS meetings will provide the foundations upon which these next generation systems will be built.

On behalf of the Organizing and Program Committees, I would like to thank the many people who helped make this conference successful. I thank all authors who submitted their work to HPCS 2007 and who are presenting in Prague. Our excellent collections of papers and presentations were possible through the diligent work of the International Technical Program Committee. The ITPC members and reviewers did an exceptional job and we are grateful for their help in reviewing and evaluating the paper submissions. We would like to acknowledge the conference three keynote speakers Prof. Pascal Berruet, RNDr. Bohdan Maslowski, DrSc., and Dr. Gabriel A. Wainer as well as our Plenary speaker Prof. Ratan K. Guha. For the first time, the

conference this year has three special sessions that were organized by Profs. Luca Spalazzi, Nouredine Melab, El-Ghazali Talbi, and Frederick C. Harris, Jr.. We are thankful for their efforts and contributions. We strongly urge all participants to organize workshops and special sessions in their area of interest in future meetings and thus grow the community. Also for the first time, the conference has a tutorial presented by Dr. Mark Wachowiak as well as a Panel Session moderated by Prof. Mads Nygård. We thank our panelists for their contributions.

The conference this year comprises of 28 out of a total of 45 papers submitted, with an acceptance rate of 62.2%. Each paper was assigned to 4-5 reviewers and the majority of authors received at least 3-4 reviews back. Due to the TPC members' timely response, we were able to meet various deadlines we had planned for the track.

We wish to thank the European Council for Modelling and Simulation members for their hard work, support, and advice, which made the conference a success. We also wish to thank our hosts at Tomas Bata University in Zlín, Czech Republic for the wonderful arrangements, support, and services they have provided. And last but not least, we thank Ms. Martina-Maria Seidel, the HPCS 2007 Conference Manager for her continual support throughout the year to make this conference possible in every way.

I must also express my gratitude for the support, guidance, and encouragement I received from our General Chairs this year. In addition, I wish to thank all members of the Organizing Committee without whom this conference and program would not have been possible.

We thank all of our attendees for making ECMS 2007 an extraordinary and enjoyable event. We hope you find this year's conference stimulating and worthwhile and look forward to seeing you at HPCS 2008.

Waleed W. Smari
HPCS 2007 Program Chair
Dayton, Ohio, USA
April 2007

HPCS 2007 PLENARY SPEECH

Wireless Network Security: Threats and Counter Measures

Ratan K. Guha
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida, USA
http://www.eecs.ucf.edu/~guha

ABSTRACT

Wireless networks (WNs) have become constitutional in businesses today. A large number of financial institutions, defence agencies, companies, and home users are using wireless technologies in their environments. However, with this unhindered mobility and flexibility lurks opportunities for violators to infringe the privacy of wireless network users. This is because the air interface is a shared medium over which data is transmitted. As companies rely on these networks for business communication and occupational activities, security threats become a major concern. There is a necessity to employ safe protocols to protect WNs from intrusion and leak of potentially sensitive information. Such protocols need to be intelligent enough to detect threats and in the advent of a breach, be robust enough to continually work against them. Thus, users should be aware of the security risks associated with the wireless networks so that they can apply appropriate measures safeguard their data/information.

In this talk, we present existing IEEE wireless standards and the efforts taken by the industry to realize these standards. IEEE 802.16 (also known as WiMAX), provides Wireless Metropolitan Area Network (WMAN) users with high-speed broadband access to the Internet, whereas, IEEE 802.11 (also known as WiFi), allows users to establish wireless connections Wireless Local Area Network (WLAN). In addition, IEEE 802.15 (also known as Bluetooth) provides short-range connectivity for portable devices.

As mentioned above, in order to capitalize the benefits of WNs, a security policy needs to be put in place to mitigate security risks. The security policy of a WN must define what is to be protected and what are the expectations of WN users. The definition of the objectives of this policy serves as a basis for security planning when new applications are designed or current networks are expanded. We consider the most common security objectives of a WN to be authentication and access control, confidentiality, integrity, and availability. The interpretation of these objectives varies as do the contexts in which they arise. These objectives aim to keep the intruders (or adversaries) at bay and allow only legitimate users to access authorized systems and services.

The standard IEEE 802.11 provided a mechanism called Wired Equivalent Privacy (WEP) to protect wireless links. WEP was a subject of criticism for many years which led to the development of IEEE 802.11i. IEEE 802.11i enhances WEP by introducing a Temporal Key Integrity Protocol (TKIP) and Counter-Mode/CBC-MAC Protocol (CCMP) with a 128-bit key to improve security. Also, it uses (Extensible Authentication Protocol (EAP) for authentication and EAP encapsulation over LANs (EAPOL) for key exchange.

Bluetooth devices establish a secure connection by using a PIN code during the initial pairing process. This PIN serves to generate a key which is used for authentication. IEEE 802.16 standard implements security in the form of a privacy sublayer, present in MAC's internal layering. Functions of the privacy sub-layer include access control and privacy of the data link.

In this talk, we present how security and privacy are fortified in all these standards.

SHORT BIOGRAPHY

Ratan K. Guha is a professor in the School of Electrical Engineering and Computer Science (www.eecs.ucf.edu) at the University of Central Florida (www.ucf.edu), Orlando. He received his B.Sc. degree with honors in Mathematics and M.Sc. degree in Applied Mathematics from University of Calcutta (www.caluniv.ac.in) and received the Ph.D. degree in Computer Science from the University of Texas at Austin (www.utexas.edu) in 1970. His research interests include distributed systems, computer networks, security protocols, modelling and simulation, and computer graphics. He has authored over 125 papers published in various computer journals, book chapters and conference proceedings. His research has been supported by grants from ARO, NSF, STRICOM, PM-TRADE, NASA, and the State of Florida. He has served as a member of the program committee of several conferences, as the general chair of CSMA'98 and CSMA'2000 and as the guest co-editor of a special issue of the Journal of Simulation Practice and Theory. He is a member of ACM, IEEE, and SCS and served as a member of the Board of Directors of SCS from 2004 to 2006. He is currently serving in the editorial board of two journals: International Journal of Internet Technology and Secured Transactions (IJITST) published by Inderscience Enterprises (www.inderscience.com), and Modelling and Simulation in Engineering published by Hindawi Publishing Corporation (www.hindawi.com).

HPCS 2007 TUTORIAL

High Performance Nonlinear Global Optimization Techniques and Applications

Mark Wachowiak
Department of Computer Science
Nipissing University, North Bay, Canada

TUTORIAL DESCRIPTION

Nonlinear global numerical optimization, wherein the best possible solution to a multi-dimensional, nonlinear model, or objective function, is sought, is an active field of study. Most useful objective functions encountered in the physical, biological, economic, and social sciences are non-smooth, non-convex, noisy, and are characterized by many local minima. For problems of very high dimensionality, the computational cost of global optimization has precluded its widespread use.

In 1995, in a seminal paper (R. B. Schnabel, "A View of the Limitations, Opportunities, and Challenges in Parallel Nonlinear Optimization", *Parallel Computing*, 21(6), 1995, pp. 875-905), three main aspects of high-performance and parallel global optimization were described: (1) Parallelizing the objective function calculation; (2) Parallelizing the underlying numerical libraries and kernels; and (3) Re-designing the algorithm for increased parallelism. The primary focus of the proposed tutorial is the third aspect, as well as completely new paradigms specifically designed for high-performance computation.

New applications of global optimization abound. For example, complex phenomena are often modelled as large systems of equations, and model parameters must be determined to correspond with experimental data. Global optimization is used to determine these parameters. Furthermore, many important engineering problems rely on simulation-based optimization, wherein the cost function itself is formed from the results of large simulation experiments. In these cases, closed-form derivatives of the objective function are generally not available, and are not easily computed. Therefore, new optimization paradigms must be considered to solve these problems.

TUTORIAL OUTLINE

The proposed tutorial would include the following topics:

- Introduction to the optimization problem and a brief overview of its classical, serial solutions.
- Parallel techniques in local and global optimization.
  - Fine-grained approaches: Parallelization of derivative computation and cost function computation.
  - Coarse-grained approaches: Searching different parts of the search space simultaneously.
- The intrinsic parallelism of deterministic global methods, including DIRECT, branch and bound, and interval analysis.
- Stochastic and computational intelligence methods, including simulated annealing, genetic algorithms, evolutionary computation, and a special emphasis on particle swarm optimization.
- Emerging computer architectures for, and applications in, high-performance global optimization.
  - Simulation-based optimization, where derivative information is not available, and the cost of computing each objective function value is very high. Important applications include safety engineering and the design of materials. High-performance derivative-free optimization methods will be discussed.
  - High-performance computing approaches to determine the optimal parameters of mathematical models that provide the best fit between observed and estimated data. Calibrating a model to observed data generally improves the model's predictive capabilities, and also provides a means for model verification and improvement.
  - Biomedical applications, particularly in imaging, computer-guided surgery and therapy, bioinformatics, and proteomics.

TARGET AUDIENCE

The target audience includes researchers, students, and practitioners who require optimization for solving large, complex problems. Specifically, those who work in simulation and modelling learn about optimization-based simulation, its implementation, and potential applications. Optimization for parameter estimation will also be discussed.

REQUIRED BACKGROUND

Although some mathematical background and some knowledge of parallel computing and computing algorithms is helpful, the tutorial will focus on applied concepts and parallelization techniques rather than on theory.

INSTRUCTOR BIOGRAPHY

Dr. Mark Wachowiak is currently an Assistant Professor at Nipissing University in North Bay, Canada. He has worked as a Postdoctoral Fellow and Research Associate at Robarts Research Institute in London, Canada, where he helped plan and build a supercomputing facility in the Imaging Laboratories. He also held an adjunct appointment in the Department of Medical Biophysics at the University of Western Ontario, London, Canada. He obtained the Doctorate degree from the University of Louisville, USA, in 2002, and was awarded the Best Dissertation Award for his work in particle swarm optimization.

Dr. Wachowiak's research interests are high-performance computing and parallel algorithms in scientific computing, grid computing, biomedical applications including imaging, bioinformatics, proteomics, and systems biology, and parallel global optimization. His recent work has focused on parallel optimization techniques for medical image alignment. He has been an invited speaker at several high-performance computing conferences.

HPCS 2007 PANEL

Parallel Processing Systems: Present and Future Trends

Moderator: Mads Nygård
Norwegian University of Science and Technology
Norway

PANEL MEMBERS:

Dr. Hamid Abachi, Monash University, Australia

Dr. Ratan Guha, University of Central Florida, US

Dr. Antonio Nebro, University of Malaga, Spain

Dr. Domenico Talia, Università della Calabria, Italy

Dr. Mark Wachowiak, Nipissing University, North Bay, Canada

ABSTRACT:

To date, a large number of research activities have taken place with particular focus on improving different aspects of parallel processing systems design including speed, reliability, fault tolerance, flexibility, compatibility, availability, cost and size.

Implementation of an appropriate interconnectivity scheme of a network is an important part of the design of parallel computer architecture. Its topology and architecture directly influence overall capability and performance of a parallel system. That is one of the reasons that we encounter many scientists and researchers continuing to develop different parallel processing architectures in order to further improve the above mentioned parameters.

This panel will discuss the current state-of-the-art in high performance parallel processing systems, the gaps that exist, and the future trends and directions of these systems.

PANLEISTS SHORT BIOS:

Hamid Abachi received his Ph.D. degree in Electrical and Computer Systems Engineering from University of Wales in Britain in 1981. He has been in academic life for more than 25 years. Hamid has also worked and gained a wide spectrum of practical experiences in heavy to light industries. From 1991 to present he has held a faculty position in the Department of Electrical and Computer Systems Engineering, at Monash University in Australia. He is the Director of the International Program, and Director of Postgraduate (Coursework) studies as well as the Professional Development Programs. He is a member of the Editorial Board of the IEEE Systems Journal in the USA and WSEAS Transactions on Computer Research. He has been a keynote speaker at many international conferences. In addition, Hamid is also a member of Technical Program Committees and a reviewer of more than 60 international conferences where in a number of occasions he has been invited to serve as the conference chair. He is a Fellow of IET (formally IEE, The Institution of Electrical Engineers, UK) and a Fellow of IEAust (Engineers Australia). Hamid is also a Senior Member of the IEEE, USA. His prime research areas include the modeling and simulation of Parallel Processing Systems, Design of Advanced Computer Architectures, Fault-tolerant Distribution and Parallel Systems. He has many journal and international conference papers in these areas.

Dr. Ratan Guha is a professor in the School of Electrical Engineering and Computer Science at the University of Central Florida. He received his B.Sc. degree with honors in Mathematics and M.Sc. degree in Applied Mathematics from University of Calcutta and received the Ph.D. degree in Computer Science from the University of Texas at Austin in 1970. He has authored over 125 papers published in various computer journals, book chapters and conference proceedings. His research has been supported by grants from ARO, NSF, STRICOM, PM-TRADE, NASA, and the State of Florida. He has served as a member of the program committee of several conferences, as the general chair of CSMA'98 and CSMA'2000 and as the guest co-editor of a special issue of the Journal of Simulation Practice and Theory. He is a member of ACM, IEEE, and SCS and served as a member of the Board of Directors of SCS from 2004 to 2006. He is currently serving in the editorial board of two journals: International Journal of Internet Technology and Secured Transactions (IJITST) published by Inderscience Enterprises, and Modelling and Simulation in Engineering published by Hindawi Publishing Corporation.

Dr. Antonio J. Nebro received his M.S. and Ph.D. degrees in Computer Science from the University of Malaga, Spain, in 1992 and 1999, respectively. He is currently an Associate Professor of Computer Science at the University of Malaga, Spain. He has coauthored several book chapters, and over 30 papers. His current research interests include the design and implementation of parallel evolutionary algorithms, multi-objective optimization, grid computing applied to meta-heuristic techniques, and applications to telecommunications and bioinformatics.

Dr. Domenico Talia is a professor at DEIS, Università della Calabria, Italy. Domenico Talia is a full professor of computer science at the Faculty of Engineering at the University of Calabria, Italy, a research associate at ICAR-CNR in Rende, Italy and a partner at Exeura s.r.l. He received the Laurea degree in Physics at University of Calabria. His research interests include grid computing, distributed knowledge discovery, parallel data mining, parallel programming languages, and peer-to-peer systems.
Dr. Talia published four books and about 200 papers in international journals such as Communications of the ACM, IEEE Computer, IEEE TKDE, IEEE TSE, IEEE TSMC-B, IEEE Micro, ACM CS, FGCS, Parallel Computing, IEEE Internet Computing and conference proceedings. He is a member of the editorial boards of the IEEE Transactions on Knowledge and Data Engineering, Future Generation Computer Systems journal, the International Journal on Web and Grid Services, the Parallel and Distributed Practices journal, and the Web Intelligence and Agent Systems International journal. He is a member of the Executive Committee of the CoreGRID Network of Excellence. He is serving as a program committee member of several conferences and is a member of the ACM and the IEEE Computer Society.

Dr. Mark Wachowiak is currently an Assistant Professor at Nipissing University in North Bay, Canada. He has worked as a Postdoctoral Fellow and Research Associate at Robarts Research Institute in London, Canada, where he helped plan and build a supercomputing facility in the Imaging Laboratories. He also held an adjunct appointment in the Department of Medical Biophysics at the University of Western Ontario, London, Canada. He obtained the Doctorate degree from the University of Louisville, USA, in 2002, and was awarded the Best Dissertation Award for his work in particle swarm optimization. Dr. Wachowiak's research interests are high-performance computing and parallel algorithms in scientific computing, grid computing, biomedical applications including imaging, bioinformatics, proteomics, and systems biology, and parallel global optimization. His recent work has focused on parallel optimization techniques for medical image alignment. He has been an invited speaker at several high-performance computing conferences.

Dr. Mads Nygård was born in Mosjøen, Norway in 1953, and he has taken both his Master of Science (Siv.Ing.) and Doctor of Science (Dr.Techn.) degrees at NTH (the Norwegian Institute of Technology, Trondheim), respectively in 1979 and 1990. From 1983 to 1997 he worked for SINTEF (the Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology, Trondheim) as Section Head, Research Manager and Principal Research Scientist in several different Information and Communication Technology departments. In 1997 he joined NTNU (the Norwegian University of Science and Technology, Trondheim) as Full Professor in the Computer and Information Science department. Over the years he has also held Adjunct Professor positions at the University of Stavanger, Stavanger (1997-2005) and the Norwegian University of Life Sciences, Ås (1994-1997). He has further for longer periods of time worked full time for UNDP (the United Nations Development Program) developing Information and Communication Technology educations in Bangkok, Thailand (1988) and Beijing, China (1984), and part time for OECD (the Organization for Economic Cooperation and Development) in a task force on Road-Vehicle Communication Systems (1989-1992). His main research interests are

distributed systems and operating systems, and he has in that capacity had longer sabbatical stays at the Imperial College in London, England (2001) and the University of Cape Town in Cape Town, South Africa (2000), and shorter sabbatical visits at the University of Maryland in Maryland College Park, USA (2002-2003) and the Georgetown University in Washington D.C., USA (2002-2003). He has more than 50 international research publications, he was the Organization Committee Chair for the Very Large Data Base 2005 conference, and he was the Panel Chairman for an International Evaluation of the Danish Computer Science Programmes in 2006. In the period 1989-1993 he was Member of the Board of Directors of SINTEF (the Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology, Trondheim), and in the period 2002-2006 he was Member of the Master of Engineering Board of NTNU (the Norwegian University of Science and Technology, Trondheim). Finally over the last 20 years he has been engaged as Chairman / Member of Several Different Boards of TEKNA (the Norwegian Society of Chartered Technical and Scientific Professionals).

TABLE OF CONTENTS

SPECIAL SESSION ON SECURITY AND HIGH PERFORMANCE COMPUTING SYSTEMS

SPECIAL SESSION ON PARALLEL AND GRID COMPUTING FOR OPTIMIZATION (PGCO 007)

SPECIAL SESSION ON HIGH PERFORMANCE INFORMATION RETRIEVAL AND VISUALIZATION: ALGORITHMS AND APPLICATIONS

IMPLEMENTATION AND EVALUATION OF CONDITIONAL STREAM IN STREAM PROCESSOR

SUI Bingcai, XING Zuocheng, MA Anguo, HUANG Ping, and ZHANG Minxuan
School of Computer Science
National University of Defense Technology, ChangSha, HuNan, China (410073)
Email: lymmeng@yahoo.com.cn

KEYWORDS

Conditional stream, stream processor, stream architecture, data-dependent control, data parallelism, data-routing

ABSTRACT

Stream processor is a new architecture designed to deal with the applications which contain abundant data-parallelisms, and it can obtain high performance for regular data-parallel applications. But if there are a few data-dependent controls in the application, its performance will be reduced very much. Conditional stream can convert data-dependent control into data-routing which can be executed in stream processors. The experimental result has shown that conditional stream can improve the performance by 2X on average at very little cost.

INTRODUCTION

Stream architecture is a new SIMD data-parallel architecture which specializes in media processing. The stream programming model partitions the application into a series of kernels, computation-intensive functions that operate on streams, and a stream program that defines the high-level control-flow and data-flow between kernels(Peter Mattson , 2002),as illustrated in Figure 1.

Figure 1: Stream Programming Model

The Imagine Stream Processor researched by Stanford University contains 8 clusters, which includes many ALUs and local register files (LRF). Similar to Imagine processor, Figure 2 shows the architecture of a SIMD stream processor with 4 clusters, which receive the same instructions from microcontroller and access their own LRFs with the same address. Existing study has shown that stream processor can provide several orders of magnitude higher performance efficiency than conventional programmable processors (Brucek Khailany, 2003).

Although the data-parallel architectures are excellent in the applications with regular data-parallelism, a simple data-dependent control can sharply reduce their performance. Because stream processors combine ideas from other architectures, the problem is potentially even more serious for stream processors (Ujval J. Kapasi, 2004).

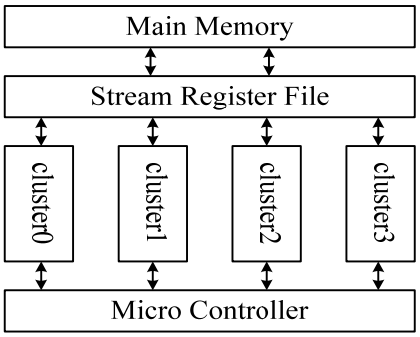


Figure 2: Stream Processor with 4 Clusters

In order to solve the problem, Ujval J. Kapasi put forward conditional stream which can convert data-dependent control into data routing, extending the application range of data-parallel architectures (Ujval J. Kapasi et al, 2000). Stream processors with conditional stream can execute the applications with data-dependent control more efficiently.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

Conditional stream is a data stream that is accessed conditionally based on the conditional codes (CC) local to the cluster (Ujval J. Kapasi et al, 2000). According to the direction of stream accessed, conditional stream can be classified into two modes:

◆ Conditional output stream

In this mode, data from one stream can be dispatched into different streams, each of which will contain homogeneous data.

During conditional output stream accessing, according to its CC, each cluster decides whether to output its result into stream buffer or not, through the communicating unit and inter-switch. If half of the double buffer is full, the data will be outputted into stream register file.

◆ Conditional input stream

In this mode, data from two or more streams which contain inhomogeneous data can be combined into one stream.

Contrary to conditional output stream, each cluster decides whether to input one data or not, through the communicating unit and inter-switch, basing on the CCS. If half of the double buffer is empty, the data will be inputted into the double buffer from stream register file.

Each processing element of conventional SIMD processors has to execute the same program code. The data-dependent conditionals are commonly implemented by mask streams. For each input element, all possible outputs are calculated and associated mask streams are generated to indicate which elements of each output stream are valid. This approach leads to some deficiencies (Ujval J. Kapasi, et al, 2000).

So in order to avoid invalid results, the processor with conditional streams routes the inhomogeneous data from input stream into two or more streams, then the stream with homogeneous data can be handled apart according to corresponding conditional clause, which can avoid invalid computing and associating mask streams since there is no invalid data existing in the stream already.

PROCESS OF CONDITIONAL STREAM

Figure 3 shows the process of an output stream in a stream processor with 8 clusters. Each cluster produces the CC needed in conditional access, according to the input data or its own computing result. In the first loop, the CCs of cluster0-cluster7 are 01110110, the bit value of which determines whether this conditional transmission is valid or not. For example, the CC of cluster0 is 0, so the result which cluster0 generates is set to be invalid, and there is no valid conditional output ongoing or valid result which is accepted by communicating unit. Contrarily, the CC of cluster1 is 1, so its result is valid, value-H, and the valid transmission is ongoing.

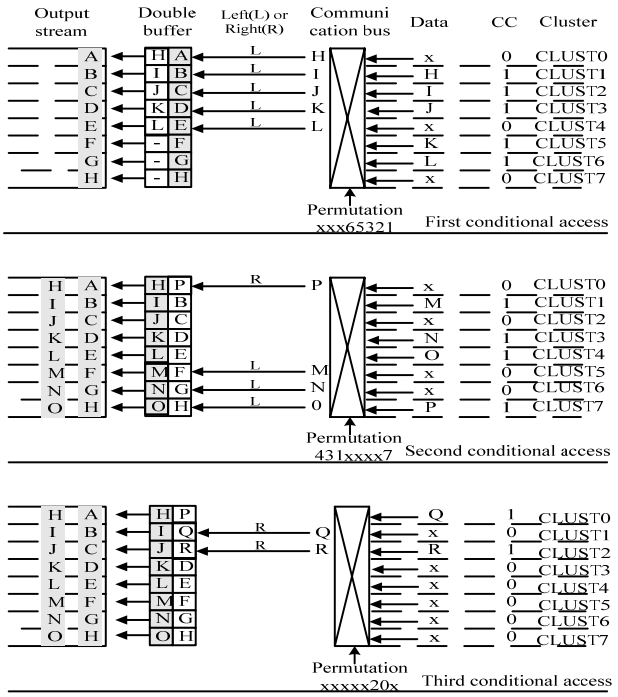


Figure 3: The Process of Conditional Stream

The number of the data outputted into stream buffer (the number of CCs) is variable. During one conditional transmission the data may not fill stream buffer to proceed stream output operation once, but the data generated during two transmissions may overflow the buffer, so a double buffer is set between the stream register file and clusters, half of which can hold all the data in a conditional transmission needed in most case, as illustrated in Figure 3.

Normal stream access is unconditional and simultaneous for all clusters, and each cluster can only access the data in the associated SB bank, for example, cluster0 can only access the first, ninth, seventeenth,…,data of stream, cluster1 can only access the second, tenth, eighteenth,…,data. But in conditional transmission, clusters must access the stream based on

the CCs to expand or compress the stream, and the entry of double buffer must be accessed orderly, therefore, the valid data must be transmitted to or from right entry through a communicating unit with full crossbar. In Figure 3, communicating unit transmits the valid data to the corresponding entry in the control of permutation, which is computed by each cluster according to the ID of the start buffer entry and the CCs of all the clusters. For example, in the first loop of Figure 3, the CCs of cluster0-cluster7 are 01110110, and the ID of start buffer entry is 0, so the permutation is xxx65321, which stands for where the data written into the entry comes from. Otherwise, it has to be pointed out into which half of the double buffer the data is written. In Figure 3, all the data of first loop are written into the left half, but only part of data in second loop are written into the left half, the remnant are written to the right. When half of buffer is full, output operation proceeds to move data to stream register file. The process of conditional input stream is contrary to conditional output stream.

IMPLEMENTATION OF CONDITIONAL STREAM

From the analysis above, it is necessary that we need a double buffer between clusters and stream register file to store data, a communicating unit to exchange data between clusters and buffer. Besides those parts, a controller is also needed to generate permutation, store the state of conditional stream and so on.

The controller can be implemented outside clusters as a whole part or inside each cluster as separate parts. In the second case, intra-switch must be implemented inside each cluster, and each controller stores their own conditional state. In other words, buffers of all the clusters have to be combined to realize the function of global buffer, each of which is equal to one entry of global buffer.

Figure 4 shows the architecture of the clusters. Each cluster can be divided into two parts: ALU and the units of conditional stream, which include COM, SP, JB and VAL. COM exchanges the data between clusters so that each cluster can receive the right data from the double buffer entry. As the double buffer, SP is used to buffer data between clusters and stream register file. JB and VAL deal with the control signals sent to COM, SP and stream register file. There are two buses outside the cluster: one for the CCs and the other for the data from clusters. In order to complete the communicating operation, each cluster must send the data others need and receive the data from another cluster.

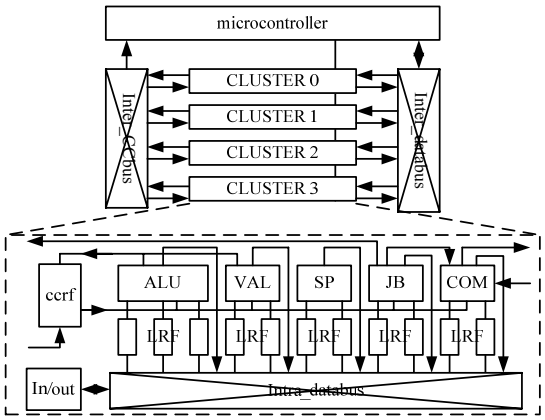


Figure 4: Architectuer of Cluster

IMPLEMENTATION OF IF-ELSE STATEMENT

There are two typical if-else statements:

- The if-else statement with two outputs

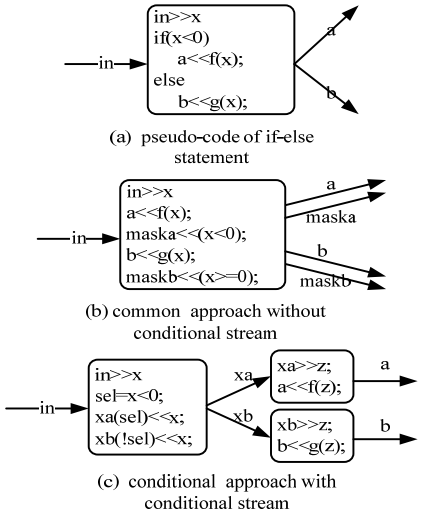


Figure 5: If-else Statement With Two Outputs

The pseudo-code of if-else statement is shown in Figure 5(a). If the data x read from input stream is minus, f(x) is computed and outputted into stream a. Otherwise, g(x) is computed and outputted into stream b.

The common approach in SIMD processor without conditional stream is shown in Figure 5(b). It partitions

the if-else statement into two kernels: kernel f and kernel g. In kernel f, all the clusters compute f(x) for all x from input stream, and store the results in the stream a. Meanwhile a mask stream (maska) is generated according to the sign of the data in order to indicate which data is valid in stream a. In kernel g, all the clusters compute g(x) for all x and output the results into stream b, another mask stream (maskb) is also generated.

The approach in SIMD stream processor with conditional stream is illustrated in Figure 5(c). The whole if-else statement is partitioned into three kernels: kernel expansion, kernel f and kernel g. In kernel expansion, all the clusters read data from input stream in and store it into different streams (xa and xb) according to the sign of the data. Then in kernel f, all the clusters computes f(x) for all minus data in stream xa and the results are outputted into the stream a. Similar to kernel f, all the non-negative data are dealt in kernel g, and the results are stored in stream b. The conditional output operation in Figure 5(c) can convert the data-dependent control into data routing, so redundant computes and invalid results are avoided to reduce the extra memory and communicating operations to handle result streams.

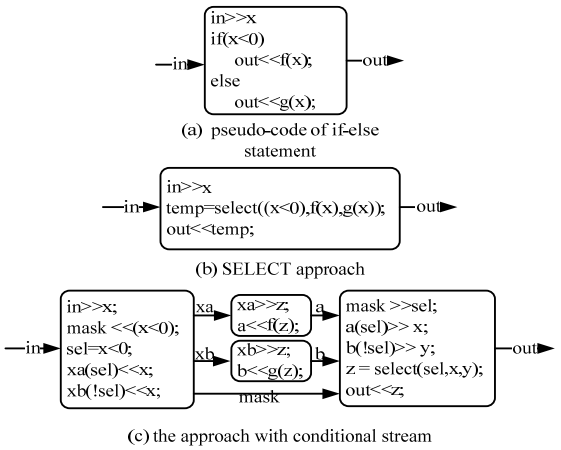


Figure 6: If-else Statement With Only One Outputs

◆ The if-else statement with one output

As shown in Figure 6(a), both f(x) for the minus data and g(x) for the others are computed and stored in stream out. But different with last sort, there is only one output stream.

The approach in SIMD processor without conditional stream is illustrated in Figure 6(b). The whole if-else statement is completed by one kernel. Each cluster computes f(x) and g(x) for each data from input stream in, then choose the right result according to the sign of the data to output stream out.

Figure 6(c) shows the approach in SIMD stream processor with conditional stream. There are four kernels in all: kernel expansion, kernel f, kernel g and kernel combination. The first three kernels are similar to the kernels in Figure 5(c). But the difference is that there is one more output stream (mask) in kernel expand, which is used in kernel combination. The conditional input operation is used in kernel combine. According to the value from the stream (mask), each cluster reads data from the stream xa or xb and outputs it into the stream out. It can be seen that the SELECT operation is also used here, but there are some nuances.

PERFORMANCE ANALYSIS OF CONDITIONAL STREAM

Now we take the if-else statement in Figure 5 as an example to analyze the performance of conditional stream. Suppose that the number of the minus is x, and the number of the non-negative data is y. Moreover we assume that m cycles are needed for computing f(x) once and n cycles for computing g(x) once.

So in Figure 5(b) it needs (x+y)*m cycles to finish computing f(x) for all the data in kernel f and (x+y)*n cycles to finish computing g(x) for all the data in kernel g. Then the total computing time of two kernels is $time_{no\_cond}=(x+y)*(m+n)$.

In the kernel f of Figure 5(c), computing f(x) for the minus needs x*m cycles and computing g(x) for the others needs y*n cycles. So the total computing time is $time_{cond}=x*m+y*n$.

Therefore for the if-else statement in Figure 5(a), compared with the approach without conditional stream, the speed-up of conditional stream is:

$$speed\_up=time_{no\_cond}/time_{cond}$$
$$=(x+y)*(m+n)/(x*m+y*n)$$
$$=1+(xn+ym)/(xm+yn)$$
$$=(1+ym/xn)/(m/n+y/x) \quad (1)$$

Assuming that the value of y/x is a, and the value of m/n is b, so in Equation (1) the speedup is:

$$speed\_up= 1+(1+ab)/(a+b). \quad (2)$$

In fact, the data are randomly distributed, so the

average value of y/x is 1, in other words, the value of a is 1. Therefore, speed_up=1+(1+ab)/(a+b) =2,we can conclude:

Conclusion 1: Compared with the approach without conditional stream, the average computing speedup of infinite distribution obtained by conditional stream is 2.

Conclusion 2: when a or b approaches 0, and b=1/a, we can get that a*b=1, a+b→∞. So (1+ab)/(a+b)→0, at this case, there is the poorest speed_up which approaches 1.

Conclusion 3: when a and b approach infinity, the speed_up =ab/(a+b) =a/2→∞, this case is the best result.

The execute time of each kernel is made up of by computing time and stall time (Sridhar Rajagopal, 2004). The first part is a large proportion. Besides, generally speaking, the computing amount in the kernel is greater, the computing proportion in the whole kernel time is larger. But above analysis only accounts for the speedup of the computing time, in fact, because there is some unavoidable overhead on memory access, the speedup of total kernel time attained by conditional stream may be less than 2.

EXPERIMENTAL RESULT

◆ Experiment 1

For the if-else statement in Figure 5, we make that $f(x)=\sin^2x+\cos^2x$, $g(x)=\sin^2x-\cos^2x$, the number of data is 1000 and the number of minus data is random. The conditional experimental result is illustrated in Figure 7(thousand cycles).

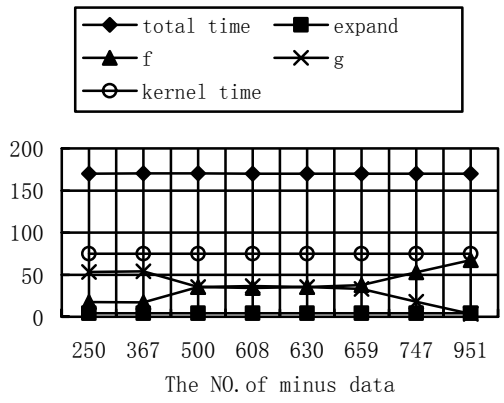


Figure 7: Execute Time of Each Kernel

From Figure 7, we conclude that the overhead of kernel expand is constant as long as the number of data is constant, and that the overhead of kernel f and kernel g change along with the number of minus data. But because the computing time of f(x) in kernel f is equal to g(x) in kernel g, $time_{cond}$ =(x+y)*m=1000*m, the whole program time does not change along with the number of minus data.

The approach in Figure 5(b) must compute f(x) for all the data and g(x) for all the data. So long as the total number of data is constant, f(x) and g(x) won't change, and the overhead of the approach without conditional stream is constant no matter how many the minus data there are. The experimental result without conditional stream is shown in Table 1(cycles).

Table 1: Spending Without Conditional Stream

Total time	256634
Kernel f	71031
Kernel g	71031
Total of kernels	142062

From the above experimental result, we can find that the speedup of kernels is 1.8X. Because m is equal to n, according to the formulate 1, the theoretical value of speedup is speed_up=1+(1+ab)/(a+b) = 2, which is independent of the number of valid data. It is because of the unavoidable memory access overhead that make the real kernel speedup a little less than 2. The total speedup is 1.51X. Because conditional stream partitions the if-else statement into several kernels, processor needs more time to load the extra program code to micro controller. Besides, kernel f and kernel g need more time to access the memory which kernel expansion writes, the total speedup is less than the speedup of kernels.

◆ Experiment 2

Table 2: Spending of Different Data Size

Data size	1000		100	
Cond or not	Y	N	Y	N
Total time	68612	146936	40966	47303
Kernel time	6057	73044	627	7344
Kernel speedup	12.1		11.7	
Total speed up	2.14		1.15	

For the if-else statement in Figure 5, make that $f(x)=\sin^2x+\cos^2x$, g(x)=x, change data size, and make the number of minus data be zero. The experimental result of

is shown in Table 2(cycles).

Again for the if-else in Figure 5, make f(x)= $\sin^2x+\cos^2x+\sin^2x-\cos^2x$, g(x)=x, change the data size, and make the number of minus data be zero, another experimental result is shown in Table 3(cycles).

Table 3: Spending of Another f(x) and g(x)

Data size	1000		100	
Cond or not	Y	N	Y	N
Total time	93280	234038	73882	76959
Kernel time	6058	127043	658	12743
Kernel speedup	20.9		19.4	
Total speedup	2.51		1.04	

◆ **Experiment 3**

For the if-else statement in Figure 6, make f(x)=$\sin^2x+\cos^2x$, g(x)=$\sin^2x-\cos^2x$, and data size is 1000, the number of minus data is random.

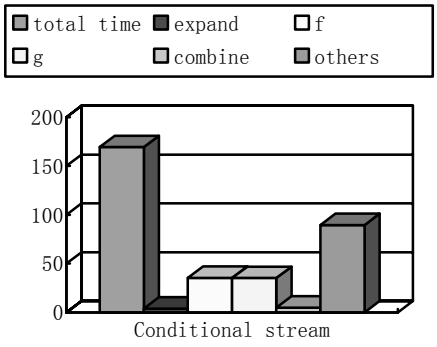


Figure 8: Experiment Result of Conditional Stream

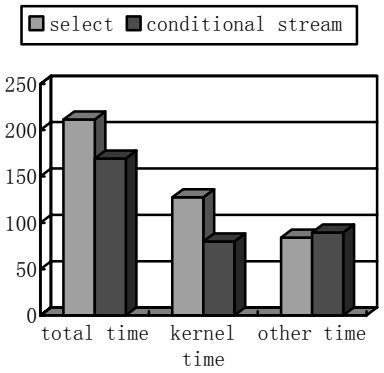


Figure 9: Comparisons of Two Approaches

Conditional stream partitions the statement into four kernels: kernel expansion, kernel f, kernel g and kernel combination. The experiment result is shown in Figure 8 and Figure 9(thousand cycles).

Of four kernels, only kernel f and kernel g execute the valid computation, and kernel expansion and kernel combination only resort the data to make computation simple and to generate right result. So the time only for computing is $timeB_{fB}+timeB_{gB}$=70778clycles, and with respect to the SELECT implement, the speedup of computing is 127531/70778=1.80; the speedup of kernel is 127531/79804=1.60; the speedup of total time is 211522/169265=1.25.

From Figure 9, we can get that the proportion of kernel time in the approach of conditional stream reduces appreciably; it is because that there are four kernels to execute and the spending on loading the program code can not be hidden.

SYNTHESIS RESULT

The X stream process with 4 clusters has been taped out at CHARTER in 130nm process. The floor plan of the processor is shown in Figure 10. The chip area is 12*12 mm^2, and it gets 34*34 mm^2 after package. The total power of the whole chip is just 8.6W.

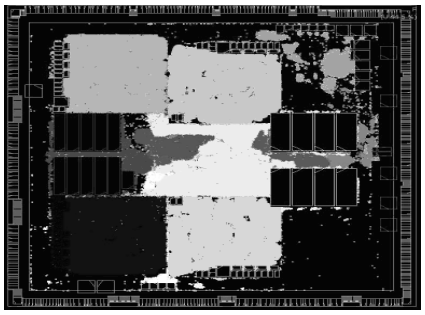


Figure 10: The Floor Plan of X Stream Processor

Table 4: Area of Conditonal Stream and Cluster

	Conditional stream	cluster
Ports	997	2128
Nets	1502	7281
Com	153292	4313977
NCom	333608	182507
Total	486895	6138413

Table 4 and Table 5 show the synthesis result of the whole cluster and conditional parts. From Table 4, we can conclude that the combination area of conditional stream units occupies 3.6% of total combination area of the whole cluster; the non-combination area occupies

18.3% of total non-combination area. The percentages of other area parameters are show in Table 4. From Table 5, we can find that the power of conditional stream only occupies a little percentage of the whole cluster, less of the whole chip.

Table 5: Power of Conditional Stream and Cluster

mw	Conditional stream	cluster
CIP	39.9687	440.5532
NSP	8.6013	19.2499
TDP	48.5700	459.8031
CLP	1.2624	20.5072

In short, conditional stream can obtain great speedup at much less cost of area, power and so on, and it is a efficient mechanism to deal with the data-dependent control.

CONCLUSIONS AND FURTHER RESEARCH

SIMD stream processor is suited to deal the applications with regular data-parallelisms, and existing studies show that it can provide more efficient performance than some conventional processor. Because stream processor combines the ideas of SIMD, VLIW, vector and so on, although stream processors can attain high performance for the data-parallel applications, the performance will slide down very much if there is only a few data-dependent controls in the applications. Conditional stream can convert data-dependent control into data routing.

Through the analysis, it shows that conditional stream can improve the performance by 2X on average in theory. Under the worst circumstances, conditional stream can still maintain the performance without any improvement, and at the best case, it can improve the performance enormously. But in fact because of memory access overhead, the average speedup may be lower than the value in theory, which can be validated through the experiments in this paper. The synthesis result shows that conditional stream only occupies a little part of the cost of the whole one cluster, and it is very efficient for stream processors.

Although conditional stream is a good mechanism, there are also some disadvantages. For example, the cluster which gets the invalid data also computes because of SIMD controlling mode. This is great waste of power, so we can stall the cluster which gets invalid data to save the power.

REFERENCES

Ujval J.Kapasi. March 2004."*Conditional Techniques for stream processing Kernels*", PhD thesis, Stanford University

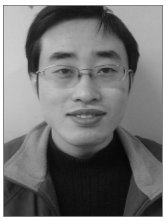
Ujval J.Kapasi, William J.Dally, Scott Rixner, Peter R. Mattson, John D. Owens, Brucek Khailany.2000. "Efficient Conditional Operations for Data-parallel Architectures", *In Proceedings of the 33rd Annual International Symposium on Micro architecture,* 159–170,Monterey,CA, ACM Press

Peter Mattson.2001."*A Programming system for the imagine media processor*", PhD thesis, Stanford University, Stanford, CA

Brucek Khailany. June 2003."*The VLSI Implementation and Evaluation of Area- and Energy-efficient Streaming Media Processors*", PhD thesis, Stanford University, Stanford, Palo Alto, CA

Sridhar Rajagopal. May, 2004."*Data-parallel Digital Signal Processors: Algorithm Mapping, Architecture Scaling and Workload Adaptation*", PhD thesis, Rice University ,Houston, TX

AUTHOR BIOGRAPHIES



SUI BING CAI was born in Shan Dong, China and went to the National University of Defense Technology, where he studied Electronic Science and Technology and obtained his bachelor degree in 2004 and master degree in 2006. Now he studies in this university for the doctor degree. His e-mail address is: lymmeng@yahoo.com.cn.

XING ZUO CHENG was born in An Hui, China. As a professor, he now works in the National University of Defense Technology and mainly researches the high performance and low-power microprocessors. He is now leading a large research group in the field of high performance computers. His e-mail address is: zcxing@nudt.edu.cn.

PHASED DROWSY I-CACHE WITH ON-DEMAND WAKEUP PREDICTION POLICY FOR HIGH-PERFORMANCE LOW-ENERGY MICROPROCESSORS

Zhou Hongwei, Zhang Chengyi and Zhang Minxuan
College of Computer Science
National University of Defense Technology
Changsha, China, 410073
E-mail: hongw.zhou@gmail.com, {chengyizhang, mxzhang}@nudt.edu.cn

KEYWORDS
Drowsy cache, Phased cache, Low energy, Instruction cache, On demand, Wakeup prediction.

ABSTRACT

In this paper, we propose a phased drowsy instruction cache with on-demand wakeup prediction policy (called "phased on-demand policy") to reduce the leakage and dynamic energy with less performance overhead. As in prior non-phased on-demand policy, an extra stage for wakeup is inserted before the fetch stage in pipeline. The drowsy cache lines are woken up in wakeup stage and the wakeup latency is overlapped with the fetch latency. Unlike in non-phased on-demand policy, The tag and data array are accessed in two phases. The tag blocks are in active mode all the time and are accessed in wakeup stage. The data blocks are in drowsy mode except when they are accessed in fetch stage. The optimum trade-off point is tried to be reached between the increment of energy caused by always active tag array in wakeup stage and reduction of energy profitted from perfect way prediction. Experiments on 9 SPCE2000 benchmarks show that, compared with prior non-phased on-demand policy, our proposed policy can save 75.4% of energy for I-Cache and improve the EDP of whole processor by 6.9%. The performance overhead is only 0.42% on average.

INTRODUCTION

Energy consumption has become the main restriction on microprocessor design because of the higher density and higher frequency. For modern microprocessors, the large capability caches are integrated in chip to improve the processors' performance. For instance, 60% of the StrongARM and 30% of Alpha 21264 are devoted to cache and memory structures (Gowan et al. 1998; Manne et al. 1998). They comprise a large portion of chip area and produce large energy. Instruction cache (I-Cache) affects total energy consumption particularly due to their high access frequency. For example, ARM920T microprocessor dissipates 25% of its total power in the I-Cache (Segars S. 2001). The energy in I-Cache consists of leakage energy and dynamic energy. According the prediction from the International Technology Roadmap for Semiconductor (SIA, 2004), the leakage energy may constitute as much as 50% of total energy by the 70nm technology. To save the energy of I-Cache at most, the leakage energy and dynamic energy in I-Cache should be reduced at the same time.

To reduce the leakage energy of I-Cache, the cache lines that are not been used recently will be put into a low-energy mode (called "drowsy" mode) and are re-woken up when accessed again. Data is not lost when the cache line is in drowsy mode. This technique is called drowsy cache (Powell et al., 2000; Kaxiras et al. 2001; Flautner et al. 2002; Chengyi Zhang et al. 2006; Kim et al. 2004a; Kim et al. 2004b; Li et al. 2004). In the drowsy mode, the supply voltage is lower than that in normal mode (called "active" mode) and data can be retained. When the drowsy cache line is accessed, the supply voltage must be recovered first. There is a wakeup penalty to restore the voltage level from the drowsy mode into the active mode. A simple policy for drowsy cache is noaccess policy in which the per-line access history is used and all the unused lines are put into drowsy mode periodically (Flautner et al. 2002). The energy saved depends on the ratio of the number of the cache lines in drowsy mode to the number of all cache lines (called "turn-off ratio"). The higher turn-off ratio is, the more leakage energy is saved.

To reduce the performance overhead caused by extra wakeup latency, the next cache line should be woken up before it is accessed. The performance overhead depends on how accurately the next cache line can be predicted and woken up. Zhang Chengyi etc. proposed a PDSR (periodically Drowsy Speculatively Recover) policy based on noaccess policy (Chengyi Zhang et al. 2006). In PDSR, a pre-wakeup mechanism is used for pre-waking up all cache lines in next sequential set when current set is being accessed. Nam Sung Kim proposed a noaccess-JITA policy in which the way predictor is also used for predicting which way may be hit in next sequential set (Kim et al. 2004a). Only the predicted cache line needs to be pre-woken up to increase turn-off ratio. The accuracy of wakeup prediction in these two policies is restricted by taken branch instruction because branch information is not used for wakeup prediction in these policies.

Sung Woo Chung etc. proposed a non-phased drowsy I-Cache with on-demand wakeup prediction policy (called "non-phased on-demand policy") in which an

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

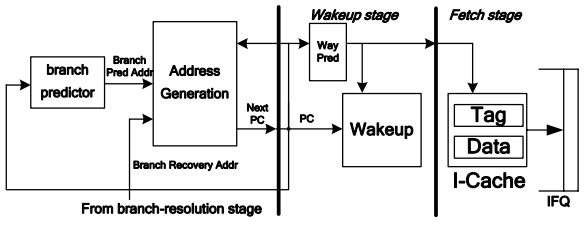
extra wakeup stage is inserted between the branch prediction and the fetch stage in pipeline (Sung Woo Chung and Kevin Skadron. 2006). The branch predictor is also used as a wakeup predictor for more accurate wakeup prediction. Extra stage hides the wakeup penalty, not affecting branch prediction accuracy. Way predictor is used for reducing the lines being woken up. All cache lines except the next expected cache line are in drowsy mode. The tag array and data array are accessed in fetch stage. The non-phased on-demand policy is near optimal policy for I-Cache leakage reduction, but it has some disadvantages yet.

In this paper, a phased drowsy instruction cache with on-demand wakeup prediction policy (called "phased on-demand policy") is proposed. In our proposed policy, the tag array is in active mode all the time and the access to tag array is moved from fetch stage to wakeup stage and the access to data array is in fetch stage as before. The cache line is accessed in two phases. The result of tag comparison is acquired one cycle earlier than non-phased on-demand policy and only the data block in matching way is necessary to be accessed. Way predictor is unnecessary. No latency caused by incorrect way prediction is incurred.

The rest of this paper is organized as follows. We first analyze the disadvantages in prior non-phased on-demand policy. Then we propose an improved phased on-demand policy. Subsequently, we introduce the performance and energy evaluation model, simulation environment and analyze the simulation results. At last, we make the conclusion and give the future work.

DISADVANTAGES IN PRIOR NON-PHASED ON-DEMAND POLICY

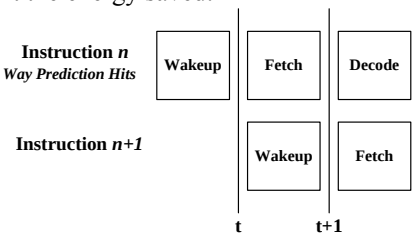
As shown in Figure 1, in non-phased on-demand policy, an extra wakeup stage for wakeup is inserted between the branch prediction and the fetch stage. Branch prediction information is also used for wakeup prediction. All cache lines are in the drowsy mode except the cache line being accessed. After the fetch-address is generated, the cache line indexed by fetch-address is woken up in wakeup stage. In fetch stage, the tag and data array of this woken cache line are both active and can be accessed within one cycle if wakeup prediction is correct.



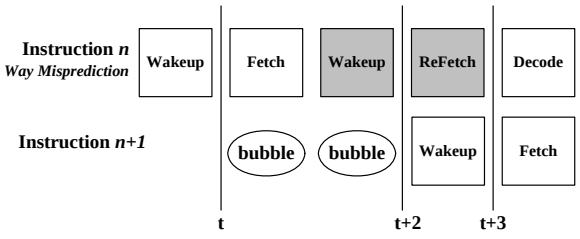
Figures 1: Front-end pipeline in Non-Phased on-demand policy

The non-phased on-demand policy is near optimal policy for reducing leakage energy of I-Cache, but two disadvantages exist and should be improved. One disadvantage is that a two-stage penalty is incurred by incorrect way prediction. This is the main reason for loss of performance.

As shown in Figure 2a, if way prediction hits, the wakeup penalty is overlapped with fetch penalty. No bubble is generated in pipeline. As shown in Figure 2b, if way prediction misses, two bubbles are generated in pipeline. One extra cycle is needed first to wake up all other cache lines in current set being accessed. Another extra cycle is necessary to access these cache lines to acquire the desired data in matching way. The performance is degraded by incorrect way prediction and additional energy for increased execution time may discount the energy saved.



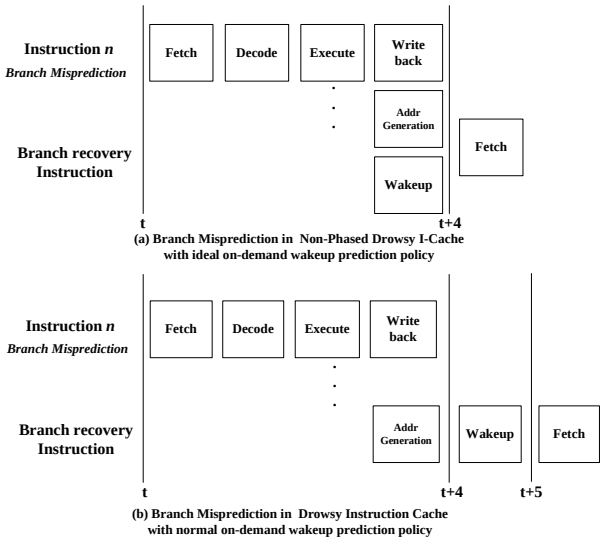
(a) Way Prediction Hits in Non-Phased Drowsy Instruction Cache with on-demand wakeup prediction policy



(b) Way Prediction Misses in Non-Phased Drowsy Instruction Cache with on-demand wakeup prediction policy

Figures 2: Pipeline comparison when Way Predictor hits and misses

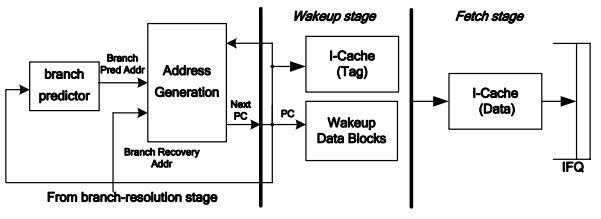
The other disadvantage is that the prior non-phased on-demand policy is ideal for recovery of branch mis-prediction. Assuming the penalty of branch mis-prediction is two cycles in conventional drowsy I-Cache. As shown in Figure 3a, in ideal non-phased on-demand policy, the address generation, branch prediction and wakeup are assumed to operate at the same time after execute/branch-resolution stage of the instruction n. So no extra wakeup penalty is incurred. The branch recovery instruction is fetched at t+4. In this ideal case, more careful adjustments are needed in pipeline design. In fact, one extra wakeup penalty will be incurred in normal non-phased on-demand policy when branch is mis-predicted, as shown in Figure 3b. The wakeup is operated after address generation. The branch recovery instruction is fetched at t+5. In this case, extra wakeup latency is incurred when branch is mis-predicted.



Figures 3: Pipeline comparison when Branch Mis-prediction

IMPROVED PHASED ON-DEMAND POLICY

As shown in Figure 4, we propose a phased drowsy instruction cache with on-demand wakeup prediction policy. The access to tag array of I-Cache is moved from fetch stage to wakeup stage. The tag and data array of I-Cache are accessed in two phases. The tag array is in the active mode all the time and only the data array can be put into drowsy mode. In each cache access, all tag blocks in current set being accessed are accessed at the same time in wakeup stage to ascertain the matching way. Then only one data block in matching way is accessed according to the result of tag comparison in fetch stage. It is the real result of tag comparison not the result of way prediction that is used to tell which way would be accessed in data array, so we can consider that the way prediction is perfect in our policy. In fact the way predictor is not used any more. Since no way prediction information can be used, so all data blocks in the set which will be accessed in fetch stage should be pre-woken up in wakeup stage with the tag comparison at the same time.



Figures 4: Phased Drowsy Instruction Cache with on-demand wakeup prediction policy

For instance, in a 4-way set-associative I-Cache with our improved policy, all four data blocks in current set being accessed are pre-woken up in advance. But in I-Cache with prior on-demand policy, only one data block is necessary to be pre-woken up if way prediction hits. So, a little more leakage energy would be consumed in data array because of high way prediction hit ratio. In addition, all the tag blocks are active at any time in our improved policy. Though the leakage energy in tag array is not saved in our proposed policy, in modern high-performance microprocessor, the number of bits in a data block is far larger than that in a tag block, so the leakage energy of tag array has less influence on the total energy of I-Cache. For example: if the data block size is 16byte and the tag address bits are 20, then the energy consumed by tag array is about 0.16 times of the energy consumed by data array.

In I-Cache with our improved policy, four tag blocks and one data block are accessed in each cache access if cache hits. No data block is accessed if cache misses. In I-Cache with prior policy, the number of tag and data blocks accessed in each cache access is determined by result of way prediction. If way prediction hits, only one tag and one data block are accessed. If way prediction misses, additional three tag and three blocks are accessed, too. Since the way prediction hit ratio is higer in prior on-demand policy, so more dynamic energy is consumed in I-Cache with our improved policy.

Comparing the prior policy, our proposed policy has less performance overhead because no extra wakeup latency is incurred by way mis-prediction. Perfect way prediction can reduce the performance overhead and save the extra energy caused by increasing execution time. That is, we use the reduction of energy profitted from perfect way prediction to balance the increment energy caused by always active tag array in wakeup stage and try to find the optimum trade-off point between them.

EVALUATION METHODOLOGY

For optimum architecture design in high-performance low-energy I-Cache, not only the leakage energy but also the dynamic energy should be considered at the same time. The additional energy in whole processor caused by increased execution time should be considered, too. This section describes the energy and performance evaluation model to evaluate our proposed policy and other policies. The simulation environment is also introduced.

Energy and Performance Evaluation Model

The base model is a conventional I-Cache without any energy controlling policy. IPC' and IPC are the number of instructions committed per-cycle with and without energy controlling policy. The program execution time is T′ or T when energy controlling policy is used or not. The normalized execution time to base model is s and can be calculated as equation (1).

$$s = T'/T = IPC/IPC' \quad (1)$$

Assuming the average proportion between energy of I-Cache (E_{icache}) and energy of whole processor (E) is α. The energy of processor except I-Cache is E_{else}. The ratio of energy in tag array (E_{icache_tag}) to data array (E_{icache_data}) is m and it can be represents proximately as the ratio of the bit width in tag array to data array. So, $E_{icache}/E=\alpha$ and $E_{icache\_tag}/E_{icache\_data}=m$.
E_d and E_s are the baseline dynamic energy and leakage energy respectively, and E'_d and E'_s are the corresponding dynamic and leakage energy with energy controlling policy.

Normalized leakage energy of I-Cache.
The e_{active} and e_{standy} represent the leakage energy of one bit memory unit during one cycle in active mode and in drowsy mode respectively. Assuming e_{standy} is q times of e_{active}, so $e_{standy}=qe_{active}$. The proportion between the number of drowsy cache lines and the number of all cache lines is $R_{turnoff}$. N_{data} and N_{tag} are the number of bits in all tag array and data array in I-Cache. In data array of I-Cache without energy controlling policy, the leakage energy is $e_{active}\ N_{data}T$. In our proposed policy, the leakage energy of drowsy data blocks is $e_{drowsy}R_{turnoff}N_{data}T'$ and the leakage energy of active data blocks is $e_{active}(1-R_{turnoff})N_{data}T'$. Equation (2) shows the normalized leakage energy to base model in data array of I-Cache.

$$E'_{s\_icache\_data}/ E_{s\_icache\_data}= [e_{active} (1-R_{turnoff}) N_{data}+e_{drowsy}R_{turnoff}N_{data}]T'/ (e_{active} N_{data}T) =[(1-R_{turnoff})+ R_{turnoff}q]s \quad (2)$$

Tag array of I-Cache is active at all the time in our proposed policy. The leakage energy of active tag blocks is increased due to longer execution time, thus, $E'_{s\_icache\_tag}=sE_{s\_icache\_tag}$. The energy caused by mode switching between the drowsy and active mode is negligible. The normalized leakage energy to base model in I-Cache is μ and calculated as shown in equation (3).

$$\mu= E'_{s\_icache} / E_{s\_icache} =(E'_{s\_icache\_tag} +E'_{s\_icache\_data})/(E_{s\_icache\_tag} +E_{s\_icache\_data}) = [m+(1-R_{turnoff})+ R_{turnoff}q]s/(m+1) \quad (3)$$

Normalized dynamic energy of I-Cache.
N_{assoc} represents the cache associativity, WPHR represents the way prediction hit ratio and CHR represents the cache hit ratio. In our proposed policy, all tag blocks in current accessed set are accessed at the same time. Only the data block in matching way is accessed when cache hits, no data block is necessary to be accessed when cache misses. The normalized dynamic energy to base model in I-Cache is ν and can be calculated as equation (4) shows.

$$\nu = E'_{d\_icache} / E_{d\_icache} =(E'_{d\_icache\_tag} +E'_{d\_icache\_data})/(E_{d\_icache\_tag} +E_{d\_icache\_data}) = (N_{assoc}\ m+CHR)/ [N_{assoc} (m+1)] \quad (4)$$

Normalized energy of else compenents in processor.
The energy of else compenernts in processor is not optimized by the energy controlling policy. In contrast, it will be increased for longer execution time. We assume that it increases in direct proportion to the execution time approximately. that is $E'_{else}=sE_{else}$.

Normalized energy of whole processor.
Not only the leakage energy but also the dynamic energy in I-Cache is considered at the same time. Assuming the ratio of leakage energy in I-Cache to leakage energy in whole processor is α_1, the ratio of dynamic energy in I-Cache to dynamic energy in whole processor is α_2 and the ratio of dynamic energy of processor to total energy in processor is β. The ratio of dynamic energy to leakage energy in I-Cache is n and can be calculated as equation (5).

$$n =E_{d\_icache}/E_{s\_icache} =(\alpha_2\beta)/[\alpha_1(1-\beta)] \quad (5)$$

So, the normalized energy of I-Cache (γ) can be calculated as equation (6) shows. The normalized energy of whole processor is calculated as equation (7) shows and it is represented as η. Equation (8) shows the normalized of EDP to base model in whole processor.

$$\gamma = (E'_{d\_icache}+E'_{s\_icache})/(E_{d\_icache} +E_{s\_icache}) = (\nu n+\mu)/ (n+1) \quad (6)$$

$$\eta = (E'_{icache}/E_{icache})\ \alpha+(E'_{else}/E_{else})(1-\alpha) =\gamma\alpha+s(1-\alpha) \quad (7)$$

$$EDP'/ EDP = (E'/E)(IPC/IPC') =\eta s \quad (8)$$

Simulation Environment

We use Hotleakage simulator (Zhang et al. 2003) to get the value of the basic parameters (IPC, IPC′, $R_{turnoff}$, and CHR) required by our evaluation modle for evaluating energy and performance. The processor parameters model a high-performance microprocessor similar to Alpha 21264 (Gowan et al. 1998) as shown in Table 1. The energy parameters are based on the 70nm/0.9V technology. Benchmarks are chosen from SPEC CPU2000. Each benchmark is first fast-forwarded a billion instructions and then simulated the 300 millions instructions. We compare our phased on-demand policy with three policies: noaccess-JITA policy, normal non-phased on-demand policy and ideal non-phased on-demand policy. These policies are described in Table2.
Because the average proportion between energy of I-Cache and energy of whole processor is different among different processors, we assume it is a moderate value (α=10%) first. We will also research other cases with different α. The parameter m and q can be calculated according to the processor parameters and energy parameters: m=0.14, q=0.04. According to the prediction from ITRS (SIA. 2004), the leakage energy will be equal to dynamic energy in microprocessors when the process technology is 70nm, so we assumes the β=0.5. According to the estimation from Hotleakage

simulator, the α_1 is 3% and α_2 is 4.3% approximately in our processor parameters model. So, n is 1.43 by equation (5). Because the increased execution time may incur extra energy, so we assume that the decay interval is 32K cycles in noaccess-JITA policy for less performance overhead.

Table 1: Architecture/Energy parameters

Processor parameters	
Instruction Window	8 IFQ, 80 RUU, 40 LSQ
Fetch/Decode/Issue/Commit width	4 instructions/cycle
L1 I-Cache	64KB, 4-way, 32B block, 1 cycle latency
Branch Predictor type	comb
Branch mis-prediction latency	2 cycles
Way prediction policy	MRU
The switch latency between different mode	1 cycle latency
Energy parameters	
Process Technology	70nm
Temperature	353K
Supply Voltage	0.9V in active mode; 0.3V in drowsy mode
Threshold Voltage	NMOS:0.1902V; PMOS: 0.2130V
Leakage energy of I-Cache in drowsy mode	0.019142 J
Leakage energy of I-Cache in active mode	0.441827 J

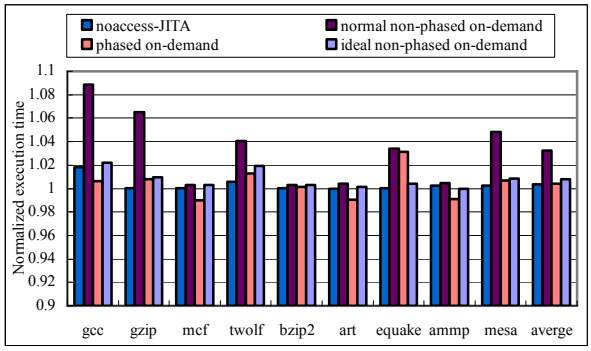
Table 2: Description for different policies

Policy	Description
noaccess-JITA	The decay interval is 32K cycles. Real way predictor is used. Branch prediction information is not used for wakeup prediction.
normal non-phased on-demand	One-cycle extra wakeup latency is incurred when branch is mis-predicted and real way predictor is used. The tag array and data array are accessed in one stage.
Ideal non-phased on-demand	No extra wakeup latency is incurred when branch is mis-predicted and a perfect way predictor is used. The tag array and data array are accessed in one stage.
Phased on-demand	One-cycle extra wakeup latency is incurred when branch is mis-predicted and way predictor is not used. The tag array and data array are accessed in two phases.

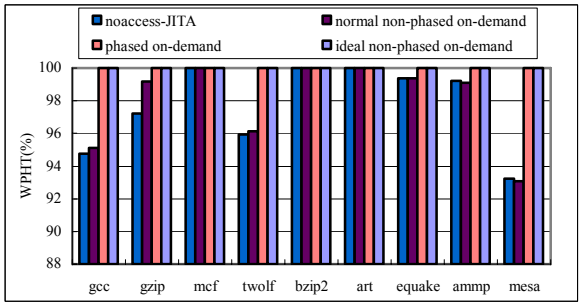
Simulation Results

As shown in Figure 5, our proposed phased on-demand policy has less performance overhead than other on-demand policies and the performance overhead is only 0.42% on average. The normalized execution time in normal non-phased on-demand policy is increased by 3.2% to base model on average. The main reason is that extra two-cycle latency is incurred by incorrect way prediction and extra one-cycle wakeup latency is incurred by incorrect branch prediction. When prefect way predictor is used and extra one-cycle wakeup latency caused by incorrect branch prediction is avoided, the performance overhead can be reduced to 0.79% as in ideal non-phased on-demand policy. The performance overhead in noaccess-JITA policy is 0.33% and it is smallest in all policies. The performance overhead in phased on-demand policy is only a little smaller than ideal non-phased on-demand policy.

The way prediction hit ratio is shown in Figure 6. Since no way predictor is used in our proposed policy, the WPHR is considered as 100% just like in ideal non-phased on-demand policy. The WPHR is 93.1% and 93.3% on average in normal non-phased on-demand policy and in noaccess-JITA policy respectively. The execution time is influenced by WPHR more obviously in normal non-phased on-demand policy. For instance, the WPHR in gcc, gzip, twolf and mesa is far lower than that in other programs, so the normalized execution time in these programs is larger than that in others. The execution time is almost not influenced by WPHR in noaccess-JITA policy.

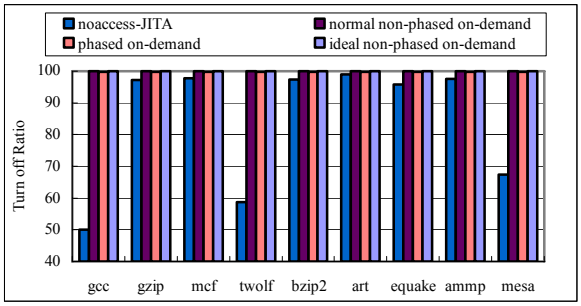


Figures 5: Normalized execution time



Figures 6: Way prediction hit Ratio

As shown in Figure 7, the turn-off ratios in all on-demand policies except noaccess-JITA policy are almost the same. In normal non-phased on-demand policy and ideal non-phased on-demand policy, the turn-off ratio is 99.95% on average and near the optimum turn-off ratio in drowsy cache. In our proposed policy, the turn-off ratio is 99.8% on average and it is only a little smaller than optimum value. In noaccess-JITA policy, the cache lines are turned off only if they are not accessed within a decay interval, so the turn-off ratio in this policy is far smaller than in other policies. It is only 67.35% on average.

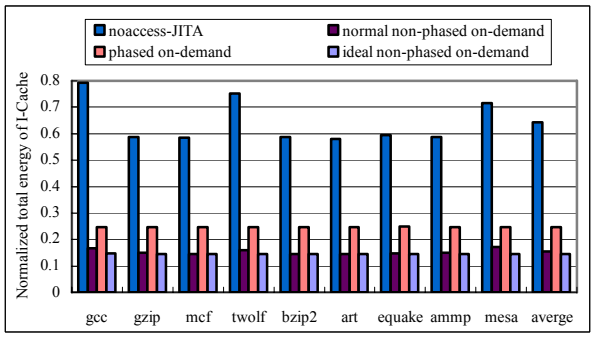


Figures 7: Turn-off ratio in drowsy I-Cache

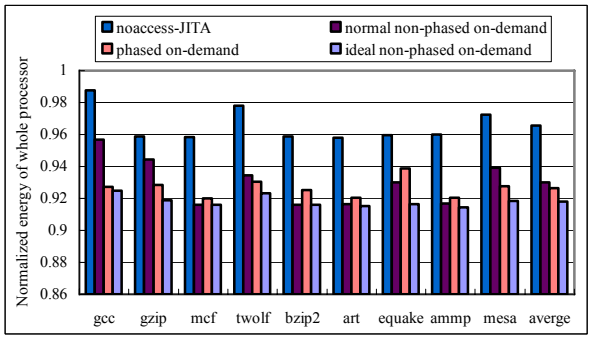
We estimate the normalized total energy of I-Cache according to our proposed energy evaluation model. As shown in Figure 8, in normal non-phased on-demand policy, the normalized total energy of I-Cache is 15.4% on average. It is very near the lowest value 14.6% in ideal non-phased on-demand policy. In our proposed policy, the normalized total energy of I-Cache is 24.6% on average. Compared with other on-demand policies, phased on-demand policy is not very good for reducing the total energy of I-Cache. However, it is better than noaccess-JITA policy. The normalized total energy of I-Cache in noaccess-JITA is 64.3% on average.

The normalized energy of whole processor is also calculated according to our proposed policy and shown in Figure 9. In our proposed policy, the normalized energy of whole processor is 92.7% on average. In ideal non-phased on-demand policy, the normalized energy of whole processor is 91.8% on average. Our proposed policy is very near the ideal non-phased on-demand policy for reducing the energy of whole processor. In normal non-phased on-demand policy, the energy saved in whole processor is less than that in our proposed policy and the normalized energy of whole processor is 93% on average. The normalized energy of whole processor in noaceess-JITA policy is 96.6% on average. It is the worst policy for reducing energy of whole processor.

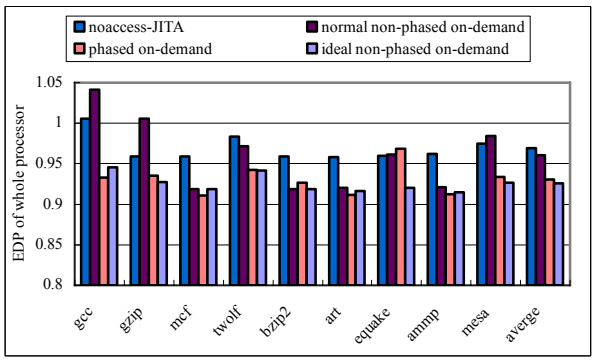
Figure 10 shows the normalized EDP of whole processor. In our proposed policy, the EDP is improved by 6.9% on average and only a little smaller than 7.4% in ideal non-phased on-demand policy. In normal non-phased on-demand policy, the EDP is only improved by 4% due to obvious performance overhead. In noaccess-JITA policy, the EDP can be improved by 3.1% on average. So, except the ideal non-phased on-demand policy, the phased on-demand policy is the best policy for improving the EDP of whole processor.



Figures 8: Normalized total energy of I-Cache



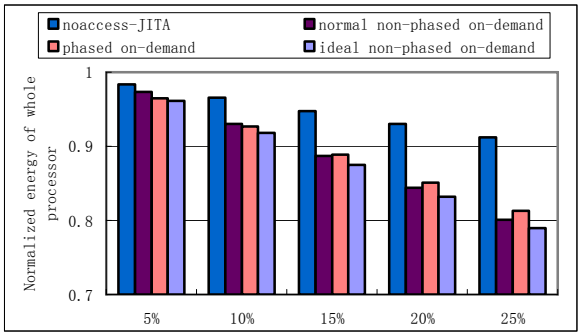
Figures 9: Normalized energy of whole processor



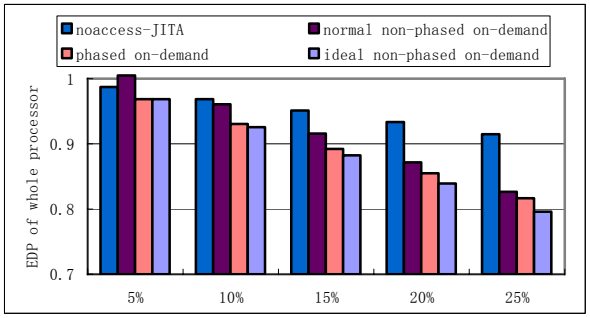
Figures 10: Normalized EDP of whole processor

As shown in Figure 11 and Figure 12, if we change the proportion of I-Cache's energy to whole processor's energy from 5% to 25% (α is increased from 5% to 25%), the energy saving and the EDP in whole processor are both improved in all policies with the increasing α. As shown in Figure 11, except the ideal non-phased on-demand policy, our proposed policy is the best policy for reducing the energy of whole processor when α is smaller than 15%. The normal non-phased on-demand policy is the best policy when α is larger than 15%. That is beause the energy saved in I-

Cache is more pivotal than the energy increased by increased execution time when the proportion of I-Cache's energy to whole processor's energy is larger. Since the performance overhead is not very obvious in I-Cache with on-demand policies, so the EDP is improved in all on-demand policies with different α. As shown in Figure 12, in all cases, the phased on-demand policy is near-ideal policy for improving the EDP in whole processor.



Figures 11: Normalized energy of whole processor



Figures 12: Normalized EDP of whole processor

CONCLUSIONS

This work presents a phased drowsy I-Cache with on-demand wakeup prediction policy. The access to tag array is moved from the fetch stage to the wakeup stage. Way predictor is not used any more. Though energy of I-Cache reduced in our proposed policy is less than that in normal non-phased on-demand policy, the performance overhead in our proposed policy is smaller and it is near the lowest performance overhead in ideal non-phased on-demand policy. In our proposed policy, the optimum trade-off point between the reduction of leakage energy in the tag array and perfect way prediction is reached. When the proportion of I-Cache's energy to whole processor's energy is 10%, with only 0.42% performance overhead, our proposed policy can reduce the energy of whole processor by 7.3% and improves the EDP by 6.9% on average. When the proportion of I-Cache's energy to whole processor's energy is increased from 5% to 25%, our proposed policy is the near-optimum policy in any cases for improving the EDP of whole processor.

REFERENCES

Chengyi Zhang, Hongwei Zhou, Minxuan Zhang, and Zuocheng Xing. 2006. "An architectural leakage power reduction method for instruction cache in ultra deep submicron microprocessors." In *the 11th Asia-Pacific Conference(ACSAC 2006),* 588-594.

Flautner K., N.S.Kim, S.Martin, D.Blaauw, and T.Mudge. 2002. "Drowsy Caches: Simple Techniques for Reducing Leakage Power." In *ISCA2002*, 147-157.

Gowan M.K., Biro L.L. and Jackson D.B. 1998. "Power Considerations in the Design of the Alpha 21264 Microprocessor." In *DAC'98*, Los Alamitos, California, U.S, 26-31.

Inoue, K., Ishihara, T., and Murakami, K. 1999. "Way-predicting Set-Associative Cache for High performance and Low Energy Consumption." In *Proc. Of 1999 International Symposium on low power Electronics and Design (ISLPED1999)*, 273-275.

Kaxiras S., Z. Hu and M. Martonosi. 2001. "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power." In *ISCA2001*, 240-251.

Kim N. S., K. Flautner, D. Blaauw and T. Mudge. 2004a. "Single-Vdd and Single-Vt Super-Drowsy Techniques for Low-Leakage High-Performance Instruction Caches." In *Proc. of Int. Symp. on Low Power Electronics and Design*, 54-57.

Kim N. S., K Flautner, D. Blaauw, and T. Mudge. 2004b. "Circuit and Microarchitectural Techniques for Reducing CacheLeakage Power." *IEEE Transaction on VLSI Systems* 12, No. 2 (Feb). 167-184.

Li Y., D. Parikh, Y. Zhang, K. Sankaranarayanan, M. Stan, and K. Skadron. 2004. "State-Preserving vs. Non-State-Preserving Leakage Control in Caches." In *Proc. of the Design Automation and Test in Europe Conference*. 22-27.

Manne S., A. Klauser, and D. Grunwald. 1998. "Pipeline Gating: Speculation Control for Energy Reduction." In *Proc. Of Int. Symp. on Computer Architecture*, 132-141.

Powell M. D. et al. 2000. "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories." In *ISLPED2000*, 90-95.

Segars S. 2001. "Low Power Design Techniques for Microprocessors." *ISSCC Tutorial*.

SIA. International Technology Roadmap for Semiconductors, 2004.

Sung Woo Chung and Kevin Skadron. 2006. "Using branch prediction information for near-optimal I-Cache leakage." In *the 11th Asia-Pacific Conference(ACSAC 2006),* 24-37.

Zhang Y., D. Parikh, K. Sankaranarayanan, K. Skadron and M. R. Stan. 2003. "Hotleakage: An Architectural, Temperature-aware Model of Subthreshold and Gate Leakage." Tech. Report CS-2003-05, Department of Computer Sciences, University of Virginia, (Mar).

Zhou Hongwei received the B.S degrees in architecture of computer from National University of Defense Technology, changsha, Hunan, P.R.China, in 2003. Currently he is working toward the Ph.D degree at National University of Defense Technology.His main research interests are architectural level power optimization for Ultra-Deep submicron microprocessors and leakage current reduction in VLSI circuits. His e-mail address is : hongw.zhou@gmail.com

PARALLEL MIN-MAX ANT COLONY SYSTEM (MMAS) FOR DYNAMIC PROCESS SCHEDULING IN DISTRIBUTED OPERATING SYSTEMS CONSIDERING LOAD BALANCING

M. Nikravan and M. H. Kashani
Department of Electrical Computer
Islamic Azad University, Shahriar Shahreqods Branch
Tehran, Iran
E-mail: moh.nikravan@gmail.com,mh.kashani@gmail.com

KEY WORDS
Distributed Systems, Scheduling, Ant Colony, Optimization, Load Balancing.

ABSTRACT

This paper presents and evaluates a new method for process scheduling in distributed systems .Scheduling in distributed operating systems has a significant role in overall system performance and throughput. An efficient scheduling is vital for system performance .The scheduling in distributed systems is known as an NP-complete problem, even in the best conditions, and methods based on heuristic search have been proposed to obtain optimal and suboptimal solutions. In this paper, we proposed an Ant-based algorithm to solve this problem considering dynamic load balancing efficiently. We evaluate the performance and efficiency of the proposed algorithm using simulation results.

INTRODUCTION

Scheduling in distributed operating systems is a critical factor in overall system efficiency. A Distributed computing system (DCS) comprising a set of Computers (Processors) connected to each other by communication networks. Process scheduling in a distributed operating system can be stated as allocating processes to processors so that total execution time will be minimized, utilization of processors will be maximized, and load balancing will be maximized. Process scheduling in a distributed system is done in two phases: in the first phase processes are distributed on computers and in the second, processes execution order on each processor must be determined. Process scheduling in distributed systems has known to be NP-complete.

Several methods have been proposed to solve scheduling problem in DCS. The proposed methods can be generally classified into three categories: Graph-theory-based approaches (Shen and Tsai 1985), mathematical models-based methods (Ma et al. 1982), and heuristic Techniques (Park 2004), (Park and Choe 2002), (Woodside and Monforton 1993), (Sarje and Sagar 1991). Heuristics can obtain suboptimal solution in ordinary situations and optimal solution in particulars. Since the scheduling problem has

Known to be NP-complete, using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms (Lin and Yang 1999), the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods (Lin and Yang 1999), (Martino 2002), (Martino and Mililotti 2003), (Moor 2003), (Oh and Wu 2004), (Wang and Korfhage 1995), (Zomaya et al. 1999) and simulated annealing (Salleh and Zomaya 1999) are considerable which extensively have been proposed in the literature.

One of the crucial aspects of the scheduling problem is load balancing. While recently created processes randomly arrive into the system, some processors may be overloaded heavily while the others are under loaded or idle. The main objectives of load balancing are to spread load on processors equally, maximizing processors utilization and minimizing total execution time (Salleh and Zomaya 1999). In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule, therefore the execution of the dynamic load balancing algorithm should not take long to arrive at a decision to make rapid process assignments. (Lan and Yu 1995) have proposed scheduling algorithms considering load balancing.

The ACO meta-heuristic was first described by Dorigo and was inspired by the ability of real ant colonies to efficiently organize the foraging behavior of the colony using external chemical *pheromone* trails acting as a means of communication. The ants deposit on the ground *pheromone* while traveling. Upon arrival at an intersection, ants make their choice of the path to follow according to a probability that is biased by the quantity of pheromones on each trail (Stutzle and Hoos 2000). Ant-based algorithms have emerged as powerful tools to solve NP-complete constrained optimization problems.

In this paper we have applied the relatively new meta-heuristic ant colony optimization (ACO) to efficiently solve this problem considering dynamic load balancing. In The proposed algorithm each ant starts with the set of

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

unscheduled processes and iteratively builds a complete solution (schedule) that shows the execution order of all existing unscheduled processes on processors. The fittest solutions are solutions which their corresponding schedules have less total execution time and communication cost, better load-balance and processor utilization. When a new solution is generated, before pheromone trails are updated a local search is applied on it to improve the quality of solution, if it is possible. We assume that the distributed system is not uniform and not preemptive, that is, the processors may be different, and a processor completes current process before executing a new one. The load-balancing mechanism used in this paper only schedule processes without process migration and is centralized.

PROBLEM DESCRIPTION AND FORMULATION

In order to schedule the processes in a distributed system, we should know the information about the input processes and distributed system itself such as: Network topology, processors speed, communication channels speed and so on. Since we study a deterministic model, a distributed system with *m* processors, *m* > 1 should be modeled as follows:

- P = $\{p_1, p_2, p_3, ..., p_m\}$ is the set of processors in the distributed system. Each processor can only execute one process at each moment, a processor completes current process before executing a new one, and a process can not be moved to another processor during execution. R is an $m \times m$ matrix, where the element r_{uv} $1 \le u, v \le m$ of R, is the communication delay rate between p_u and p_v. H is an $m \times m$ matrix, where the element h_{uv} $1 \le u, v \le m$ of H, is the time required to transmit a unit of data from p_u to p_v. It is obvious that $h_{uu} = 0$ and $r_{uu} = 0$.
- $T = \{t_1, t_2, t_3, ..., t_n\}$ is the set of processes to execute. A is an $n \times m$ matrix, where the element a_{ij} $1 \le i \le n$, $1 \le j \le m$ of A, is the execution time of process t_i on processor p_j .In homogeneous distributed systems the execution time of an individual process t_i on all processors is equal, that means : $1 \le i \le n$ $a_{i1} = a_{i2} = ... = a_{im}$. D is a linear matrix, where the element d_i $1 \le i \le n$ of D, is the data volume for process t_i to be transmitted, when process t_i is to be executed on a remote processor.
- F is a linear matrix, where the element f_i $1 \le i \le n$ of F, is the target processor that is selected for process t_i to be executed on. C is a linear matrix, where the element c_i $1 \le i \le n$ of C, is the processor that the process t_i is presented on just now.

The problem of process scheduling is to assign for each process $t_i \in T$ a processor $f_i \in P$ so that total execution time will be minimized, utilization of processors will be maximized, and load balancing will be maximized. In such systems there are finite numbers of processes, each having a process number and a execution time and placed in a process pool from which processes are assigned to processors. The main objective is to find a schedule with minimum cost. The following definitions are also needed:

Definition 1

The processor load for each processor is the sum of processes execution times allocated to that processor. However, as the processors may not always be idle when a schedule is evaluated, the current existing load on individual processors must also be taken into account, therefore (1):

$$Load(p_i) = \overset{\text{No. of allocated processes on processor } i}{\sum_{j=1} a_{j,i}} + \overset{\text{No. of New Assigned processes to processor. } i}{\sum_{k=1} a_{k,i}} \quad (1)$$

Definition 2

The length or *make span* of a schedule T is the maximal finishing time of all processes or maximum load. Also communication cost (CC) to spread recently created processes on processors and the Completion Time (ct) of a process t_i on processor p_j must be computed (2,3,4):

$$\max span(T) = \max\left(Load(p_i)\right) \quad \forall 1 \le i \le Number\ of\ \text{Pr}ocessors \quad (2)$$

$$CC(T) = \overset{number\ of\ new\ processes}{\sum_{i=1}} \left(r_{c_i f_i} + h_{c_i f_i} \times d_i\right) \quad (3)$$

$$ct_{ij} = a_{ij} + \overset{\text{number of processes in processor j's queue}}{\sum_{k=1} a_{kj}} \quad (4)$$

Definition 3

The Processor utilization for each processor is obtained by dividing sum of processing times by maxspan, and the Average of processors utilization is obtained by dividing sum of all utilization by number of processors (5, 6):

$$U(p_i) = Load(p_i) \Big/ \max span \quad (5)$$

$$AveU = \left(\sum_{i=1}^{No\ of\ processors} U(p_i)\right) \Big/ Number\ Of\ \text{Pr}ocessors \quad (6)$$

Definition 4

Number of Acceptable Processor Queues (NoAPQ): We must define thresholds for light and heavy load on processors. If the processes completion time of a processor (by adding the current system load and those contributed by the new processes) is within the light and heavy thresholds, this processor queue will be acceptable. If it is above the heavy threshold or below the light-threshold, then it is unacceptable, but what is important is average of number of acceptable processors queues, which is achievable by (7):

$$AveNoAPQ = NoAPQ / Number\ Of\ \text{Pr}ocessors \quad (7)$$

Definition 5

A Queue associated with every processor, shows the processes that processor has to execute.

THE PROPOSED ANT-BASED ALGORITHM

Pheromone Trail Defining

As discussed before, the process scheduling problem can be stated as allocating processes on processors in a way that satisfies the mentioned objectives. Therefore the pheromone value $\tau_k(t_i, p_j)$ was selected to represent the desirability of allocating process t_i on processor p_j in *k*' th iteration. We define a process-processor pheromone trail matrix which each process-processor pair has a single entry in the matrix. In MMAS we initialize the pheromone trails in such a way that after the first iteration all pheromone trails correspond to $\tau_{\max}(1)$. This can easily be achieved by initially setting τ_0 to some arbitrarily high value. After the first iteration of MMAS, the trails will be forced to take values within the imposed bounds; in particular, they will be set to $\tau_{\max}(1)$. This type of trail initialization is chosen to increase the exploration of solutions during the first iterations of the algorithm.

The Heuristic

The heuristic used here to select the next process to be scheduled is a modification to what is presented in (Sarje and Sagar 1991); this heuristic is called 'Relative Cost' (RC). According to our objectives in scheduling, it is better to consider two factors, when a process is allocated to a processor. First, matching, that means a process should be allocated to a processor that will completes it fastest possible. Second, load-balancing that means load should be balanced over all processors. In order to take into account both these factors … RC method defines to measures, *static relative cost (SRC) and dynamic relative cost (DRC)*. The SRC of a job processor pair (t_i, p_j) is simply the execution time of process t_i on processor p_j divided by the average execution time of process t_i on all processors, as shown in (8), and is fixed for the entire run. The DRC is re-calculated after each process is scheduled, and is the completion time, *c*t, of each process t_i on processor p_j, divided by the average *ct* of t_i on all processors, shown in (9).

$$SRC(t_i, p_j) = \frac{a_{ij}}{\left(\sum_{k=1}^{m} a_{ik}\right)/m} \quad (8)$$

$$DRC(t_i, p_j) = \frac{ct_{ij}}{\left(\sum_{k=1}^{m} ct_{ik}\right)/m} \quad (9)$$

The DRC and SRC and The best suitable processor are calculated for each unscheduled process each iteration. The best suitable processor for a process t_i is the processor p_j that maximizes SRC and DRC … and obtained as follows:

$$p_{best}(t_i) = \max_{\forall p_j}((\alpha \times SRC(t_i, p_j)) \times (\beta \times DRC(t_i, p_j))) \quad (10)$$

The heuristic use by the ants for each t_i is obtained as follows:

$$\eta(t_i) = (\alpha \times SRC(t_i, p_{best}(t_i))) \times (\beta \times DRC(t_i, p_{best}(t_i))) \quad (11)$$

Where these equations α and β are parameters used to control the effect of each value.

Fitness Function

As discussed before, the main objective of scheduling is to find a schedule with optimal cost while load balancing, processors utilization and cost of communication are considered. We take into account all objectives in following equation. The fitness of a Schedule T (12):

$$fitness(T) = \frac{(\gamma \times AveU) \times (\theta \times AveNoAPQ)}{(\alpha \times \max\ span(T)) \times (\beta \times CC(T))} \quad (12)$$

Which $0 < \alpha, \beta, \gamma, \theta \le 1$ are control parameters to control effect of each part according to special cases and their default value is one. This equation shows that a fitter solution (Schedule) has less make span, less communication cost, higher processor utilization and higher Average number of acceptable processor queues.

Updating The Pheromone Trail

As discussed before In MMAS only one single ant is used to update the pheromone trails in each iteration. Therefore, pheromone trails are updated as follows:

$$\tau_{k+1}(t_i, p_j) = \begin{cases} \rho \times \tau_k(t_i, p_j) + \Delta & \text{if } t_i \text{ is allocated on} \\ & \text{processor } p_j \text{ in } s^{ib} \\ \rho \times \tau_k(t_i, p_j) & \text{otherwise} \end{cases}$$

Where $\Delta = \frac{f(s^{ib})}{f(s^{gb})}$ and $f(s)$ is the cost of solution *S*. (s^{ib}) Is the *iteration-best* solution and (s^{gb}) is the *global-best* solution. It is to be noted that when a better solution than the *global-best* solution is found, the *global-best* solution is set to this solution. In MMAS search stagnation may occur. This can happen if at each choice point, the pheromone trail is significantly higher for one choice than for all the others. In such a situation the ants construct the same solution over and over again and the exploration of the search space stops. Obviously, such a stagnation situation should be avoided. By limiting the influence of the pheromone trails one can easily avoid the relative differences between the pheromone trails from becoming too extreme during the run of the algorithm. To achieve this goal, MMAS imposes explicit limits $\tau_{\min}$ and $\tau_{\max}$ on the minimum and maximum pheromone trails such that for all pheromone trails $\tau_k(t_i, p_j)$, $\tau_{\min} \le \tau_k(t_i, p_j) \le \tau_{\max}$. So after each pheromone updating if $\tau_k(t_i, p_j) < \tau_{\min}$ then we set $\tau_k(t_i, p_j) = \tau_{\min}$, and if $\tau_k(t_i, p_j) > \tau_{\max}$ then we set $\tau_k(t_i, p_j) = \tau_{\max}$. It is to be noted that always $\tau_{\min} > 0$.

Building a Solution

Each ant starts with an empty solution, and iteratively adds components to the solution until the solution is completed (all processes are scheduled).each iteration the next process to be scheduled is probabilistically selected according to heuristic value and pheromone trial information, and then the selected process is allocated to its best processor. The probability of process t_i to be selected is obtained as follows (13):

$$probability(t_i) = \frac{[\tau(t_i, p_{best}(t_i))]^{\alpha}.[\eta(t_i)]^{\beta}}{\sum_{\substack{\forall unscheduled \\ task \quad t_k}} [\tau(t_k, p_{best}(t_k))]^{\alpha}.[\eta(t_k)]^{\beta}} \quad (13)$$

Where these equations α and β are parameters used to control the effect of each value.

Local Search

When a complete solution is made by an ant before updating pheromone trails, a local search is applied on the solution to improve its quality, if it is possible. The local search is performed on every ant, every iteration, so it needs to be fairly fast. A simple approach is to check if any jobs could be swapped between processors which would result in a lower make span.

Mutation

In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule, therefore the execution of the dynamic load balancing algorithm should not take long to arrive at a decision to make rapid processes assignments. As ACO algorithms are population-based, each time that the algorithm has to schedule a new set of processes, a large amount of time may be consumed to construct ant colony and then ants' sequentially construct their solutions. Therefore a parallel version of this algorithm is more efficient. To run the ACO algorithm in parallel, we propose two models.

Message Passing Model

In this model we have one *Master* ant, and a number of *subordinate* ants. The master works as a coordinator. It sets the parameters and initializes the pheromone trails. Each subordinate ant is placed on an individual processor and is run on it. In start of each iteration the master sends the pheromone trail matrix to all subordinates, then building solutions by subordinate ants is done in parallel, finally solutions are returned to master. Master updates pheromone trails using solutions; this cycle is repeated until termination condition is met

Island Model

The island model is inspired from GA theory. In the island model, the whole colony is divided to a number of sub colonies. Each sub colony is placed on an individual processor. All colonies work in parallel after a defined number of iterations the colonies interchange the best local solutions they found. When a colony receives a solution that is better than its best solution, its best solution is set to the solution which received, and pheromone trails are updated according to new solution.

Termination Condition

We can apply multiple choices for termination condition. Max number of generation, algorithm convergence, equal fitness for fittest selected in respective iterations.

The Structure of Proposed GA-Based Algorithm

Our proposed Ant Colony-Based algorithm starts with a fixed number of ants. An individual ant constructs candidate solutions by starting with an empty solution and then iteratively adding solution components until a complete candidate solution is generated. A certain

fitness function is used to evaluate the fitness of each solution. After the solution construction is completed, the ants give feedback on the solutions they have constructed by depositing pheromone on solution components which they have used in their solution. Typically, solution components which are part of better solutions or are used by many ants will receive a higher amount of pheromone, and hence, will more likely be used by the ants in future iterations of the algorithm. This process iterates until termination condition is satisfied, Figure 1.

Procedure *ANT Scheduler*
Begin
Set *parameters and initial pheromone trails;*
While *(termination condition not met)* ***do***
For i=1 to NOANTS do
Cons. a Schedule S_i and *Local Search to improve* S_i ;
End For*;*

$$s^{ib} = \left\{ s_j : f(s_j) = \max_{i=1}^{NOANTS} (f(s_i)) \right\};$$

Update *Trails Using* s^{gb} *and* s^{ib} ;

if $f(s^{ib}) > f(s^{gb})$ ***then*** $f(s^{gb}) = f(s^{ib});$
End While;
Return *The best solution found* s^{gb} ;
End

Figure 1: The Structure Of Proposed Algorithm

EXPERIMENTAL RESULTS

In this section, we have used the simulation results to show the performance of the proposed ACO based algorithm. Current solution techniques are concentrated on scheduling tasks with precedence constraints so our approach is not completely comparable with them. The parameter values used in ACO algorithms are often very important in getting good results, however the exact values are very often entirely problem dependent , and cannot always be derived from features of the problem itself. We have tried different values of the population size (*POPSIZE*), the extent to which pheromone information is used (α), the extent to which heuristic information is used (β), to find which values would steer the search towards the best solution. The best result achieved is as follows: $\alpha = 2$, $\beta = 20$, *NOANTS*=10,*NOGEN* =50, *m*=10 (number of processors), *n=100...900* (number of processes). Measurement of performance of these algorithms was based on three metrics: total completion time and average processor utilization. The default parameters were varied and the results collected from test runs were used to study the effects of changing these parameters.

Changing The Number of Processes

We have studied the effect of increasing number of processes on total completion time and average processor utilization. The Obtained results are shown in Figure 2.and, Figure 3. A considerable point in Figure 3 is that when number of processes is increased, higher utilization is obtained.

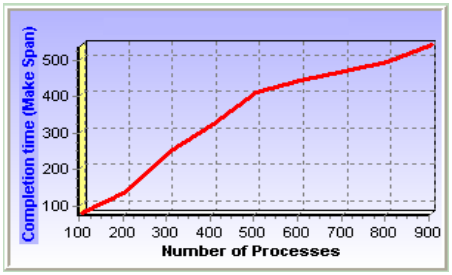


Figure 2: Total Completion Time

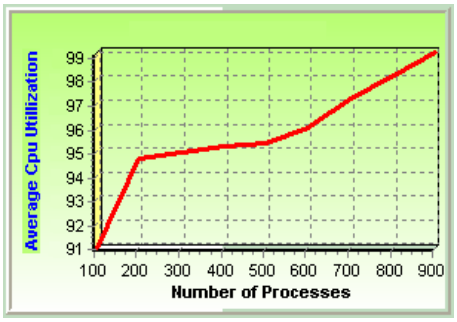


Figure 3: Average Processor Utilization

Changing The Number of Generations

When the number of generations was increased our proposed algorithm had a better function. The Obtained results are shown in Figure 4 and, Figure 5. While the number of generations was increased the total completion time was reduced, it is because that the components of fitter solutions are selected by more ants, so the amount of deposited pheromone on these components reinforced and, therefore they more likely will be selected in next generations. The result is that the quality of the generated process assignment improves after each generation. A considerable point in these figures is that when the number of generations was increased, due to its high convergence, higher utilization is obtained.

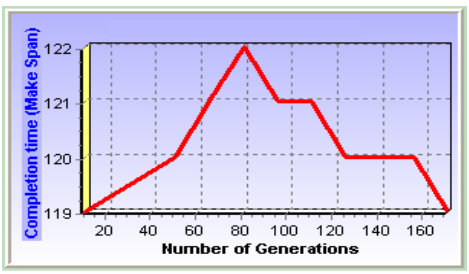


Figure 4: Total Completion Time

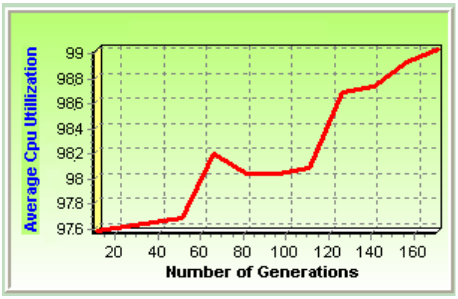


Figure 5: Average Processor Utilization

Changing The Size of Population

Changing the size of population is also considerable in terms of total completion time, processor utilization. The Obtained results are shown in Figure 6 and, Figure 7. While the size of population was increased the total completion time was decreased and, average processor utilization was increased. This is because that while the number of ants increased the search space of solutions exploited better, and it leads to better schedules.

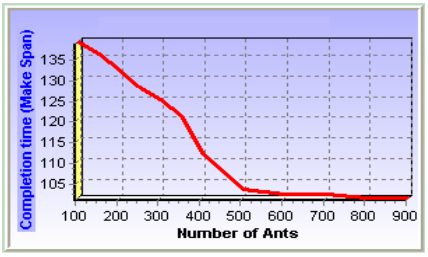


Figure 6: Total Completion Time

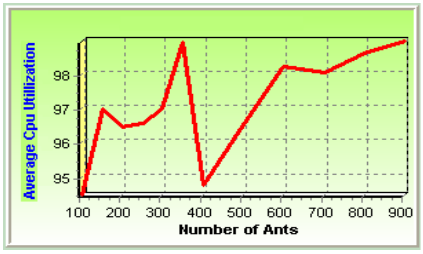


Figure 7: Average Processor Utilization

CONCLUTIONS

Scheduling in distributed operating systems has a significant role in overall system performance and throughput. We have presented and evaluated a relatively new meta-heuristic ant colony optimization method to solve this problem. This algorithm considers multi objectives in its solution evaluation and solves the scheduling problem in a way that simultaneously minimizes, makes span, and maximizes average processor utilization and load-balance.

REFERENCES

Cheng-Fa, Tsai and Chun-Wei Tsai. 2002. ”A new approach for solving large traveling salesman problem using evolutionary ant rules”. *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, pp. 1540-1545.

Lan , Y. and T. Yu. 1995. " A Dynamic Central Scheduler Load-Balancing Mechanism". *Proc. IEEE 14th Ann. Int'l Phoenix Conf. Computers and Comm.*, pp. 734-740.

Lin, M. and L.T. Yang. 1999. " Hybrid Genetic Algorithms for Scheduling Partially Ordered Tasks in a Multi-processor Environment". *In Proceedings of the 6th International Conference on Real-Time Computer Systems and Applications*, pp. 382 –387.

Ma, P.Y.R.; E.Y.S. Lee; and J. Tsuchiya. 1982. "A Task Allocation Model for Distributed Computing Systems". *IEEE Trans. On Computers*, Vol. 31, No. 1, pp. 41-47.

Martino, V. D. and M. Mililotti. 2002. " Scheduling in a Grid computing environment using Genetic Algorithms". *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium.*

Martino, V. D. 2003. "Sub Optimal Scheduling in a Grid using Genetic Algorithms". *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium.*

Moor, M. 2003. "An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster". *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium.*

Oh, J. and C. Wu. 2004. "Genetic-algorithm-based Real-time Task Scheduling With Multiple Goals". *International Journal of Systems and Software*, Vol. 71, pp. 245-258.

Park, C.I. and T.Y. Choe. 2002. "An optimal scheduling algorithm based on task duplication". *IEEE Transactions on Computers*, Vol. 51 ,No. 4 ,pp. 444–448.

Park, G.L. 2004. "Performance Evaluation of a List Scheduling Algorithm In Distributed Memory Multiprocessor Systems", *Journal of Future Generation Computer Systems*, Vol. 20, pp. 249-256.

Salleh, S. and A.Y. Zomaya. 1999. " Scheduling in Parallel Computing Systems: Fuzzy and Annealing Techniques", *Kluwer Academic*.

Sarje, A.K. and G. Sagar. 1991. "Heuristic Model for Task Allocation in Distributed Computer Systems", *Proceedings of the IEEE* , Vol. 138, No. 5, pp. 313-318.

Shen, C. C. and W.H. Tsai. 1985. "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Using a Minimax Criterion ". *IEEE Trans. On Computers*, Vol. 34, No. 3, pp. 197-203.

Solnon, C. 2002. "Ants Can Solve Constraint Satisfication Problems". *IEEE Trans. On Evolutionary Computation*, Vol. 6, No. 4, pp. 347-357.

Stutzle, T. and H. H. Hoos. 2000. "Max-Min Ant System". *Journal of Future Generation Computer Systems*, Vol. 16, pp. 889-914.

Wang, P. and W. Korfhage. 1995. ”Process Scheduling Using Genetic Algorithms”. *IEEE. Symposium on Parallel and Distributed Processing*, pp. 638-641.

Woodside, C.M. and G.G. Monforton. 1993. "Fast Allocation of Processes in Distributed and Parallel Systems". *IEEE Trans. On Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 164-174.

Zomaya, A. Y.; C. Ward; and B. Macey. 1999. " Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues ". *IEEE Trans. On Parallel and Distributed Systems* , Vol. 10, No. 8, pp. 795-812.

Zomaya, A. Y. ,and Y. Teh. 2001. "Observations on Using Genetic Algorithms for Dynamic Load-Balancing ". *IEEE Trans. On Parallel and Distributed Systems*, Vol. 12, No. 9, pp. 899-911.

A GENETIC ALGORITHM FOR PROCESS SCHEDULING IN DISTRIBUTED OPERATING SYSTEMS CONSIDERING LOAD BALANCING

M. Nikravan and M. H. Kashani
Department of Electrical Computer
Islamic Azad University, Shahriar Shahreqods Branch
Tehran, Iran
E-mail: moh.nikravan@gmail.com,mh.kashani@gmail.com

KEY WORDS
Distributed systems, scheduling, genetic algorithm, simulated annealing , load balancing.

ABSTRACT

This paper presents and evaluates a new method for process scheduling in distributed systems. Scheduling in distributed operating systems has a significant role in overall system performance and throughput. An efficient scheduling is vital for system performance. The scheduling in distributed systems is known as an NP-complete problem even in the best conditions, and methods based on heuristic search have been proposed to obtain optimal and suboptimal solutions. In this paper, using the power of genetic algorithms we solve this problem considering load balancing efficiently. We evaluate the performance and efficiency of the proposed algorithm using simulation results.

INTRODUCTION

Scheduling in distributed operating systems is a critical factor in overall system efficiency. A Distributed Computing system (DCS) is comprised of a set of Computers (Processors) connected to each other by communication networks. Process scheduling in a distributed operating system can be stated as allocating processes to processors so that total execution time will be minimized, utilization of processors will be maximized, and load balancing will be maximized. Process scheduling in a distributed system is done in two phases: in the first phase processes are distributed on computers, and in the second processes execution order on each processor must be determined .Process scheduling in distributed systems has been known to be NP-complete.

Several methods have been proposed to solve scheduling problem in DCS. The proposed methods can be generally classified into three categories: Graph-theory-based approaches [23], mathematical models-based methods [24], and heuristic Techniques [2, 6, 18, 21].

Heuristics can obtain suboptimal solution in ordinary situations and optimal solution in particulars. Since the scheduling problem has been known to be NP-complete, using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms [13],the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods [1,4,5,11,13,14,17] and simulated annealing [12,20] are considerable which extensively have been proposed in the literature.

One of the crucial aspects of the scheduling problem is load balancing. While recently created processes randomly arrive into the system, some processors may be overloaded heavily while the others are under-loaded or idle. The main objectives of load balancing are to spread load on processors equally, maximizing processors utilization and minimizing total execution time [12]. In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule, therefore the execution of the dynamic load balancing algorithm should not take long to arrive at a decision to make rapid task assignments. [9,15,16,19,22] have proposed scheduling algorithms considering load balancing .

A GA starts with a generation of individuals, which are encoded as strings known as chromosomes. A chromosome corresponds to a solution to the problem. A certain fitness function is used to evaluate the fitness of each individual. Good individuals survive after selection according to the fitness of individuals. Then the survived individuals reproduce offspring through crossover and mutation operators. This process iterates until termination condition is satisfied [10]. GA-based algorithms have emerged as powerful tools to solve NP-complete constrained optimization problems, such as traveling salesman problem, job-shop scheduling and flow-shop scheduling, machine learning, VLSI technology, genetic synthesis and etc [3, 7].

In this paper using the power of genetic algorithms we solve this problem considering load balancing efficiently. The proposed algorithm maps each schedule with a chromosome that shows the execution order of all existing processes on processors. The fittest chromosomes are selected to reproduce offspring; chromosomes which their corresponding schedules have less total execution time, better load-balance and processor utilization. We assume that the distributed system is non-uniform and non-preemptive, that is, the processors may be different, and a processor completes current process before executing a new one. The load-

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

balancing mechanism used in this paper only schedule processes without process migration and is centralized.
The remainder of this paper is organized as follows: The problem description and formulation is given in Section 2, In Section 3, we describe the proposed algorithm. Section 4, gives the performance evaluation of the proposed algorithm in comparison with other similar algorithms. Section 5 concludes this research.

PROBLEM DESCRIPTION AND FORMULATION

In order to schedule the processes in a distributed system, we should know the information about the input processes and distributed system itself such as : Network topology, processors speed, communication channels speed and so on. Since we study a deterministic model, a distributed system with *m* processors, *m* > 1 should be modeled as follows:

- P = $\{p_1, p_2, p_3, ..., p_m\}$ is the set of processors in the distributed system. Each processor can only execute one process at each moment, a processor completes current process before executing a new one, and a process can not be moved to another processor during execution. R is an $m \times m$ matrix, where the element r_{uv} $1 \le u, v \le m$ of R, is the communication delay rate between p_u and p_v. H is an $m \times m$ matrix, where the element h_{uv} $1 \le u, v \le m$ of H, is the time required to transmit a unit of data from p_u to p_v. It is obvious that $h_{uu} = 0$ and $r_{uu} = 0$.
- $T = \{t_1, t_2, t_3, ..., t_n\}$ is the set of processes to execute. A is an $n \times m$ matrix, where the element a_{ij} $1 \le i \le n$, $1 \le j \le m$ of A, is the execution time of process t_i on processor p_j .In homogeneous distributed systems the execution time of an individual process t_i on all processors is equal, that means : $1 \le i \le n$ $a_{i1} = a_{i2} = ... = a_{im}$. D is a linear matrix, where the element d_i $1 \le i \le n$ of D, is the data volume for process t_i to be transmitted, when process t_i is to be executed on a remote processor.
- F is a linear matrix, where the element f_i $1 \le i \le n$ of F, is the target processor that is selected for process t_i to be executed on. C is a linear matrix, where the element c_i $1 \le i \le n$ of C, is the processor that the process t_i is presented on just now.

The problem of process scheduling is to assign for each process $t_i \in T$ a processor $f_i \in P$ so that total execution time will be minimized, utilization of processors will be maximized, and load balancing will be maximized. In such systems there are finite numbers of processes, each having a process number and a execution time and placed in a process pool from which processes are assigned to processors. The main objective is to find a schedule with minimum cost. The following definitions are also needed:

Definition 1

The processor load for each processor is the sum of processes execution times allocated to that processor. However, as the processors may not always be idle when a chromosome (schedule) is evaluated, the current existing load on individual processors must also be taken into account, therefore (1):

$$Load(p_i) = \sum_{j=1}^{\substack{No.\ of\ allocated\ processes\ on\ processor\ i}} a_{j,i} + \sum_{k=1}^{\substack{No.\ of\ New\ Assigned\ processes\ to\ processor.\ i}} a_{k,i} \qquad (1)$$

Definition 2

The length or *maxspan* of a schedule T is the maximal finishing time of all processes or maximum load. Also communication cost (CC) to spread recently created processes on processors must be computed (2,3) :

$$\max span(T) = \max\left(Load(p_i)\right) \quad \forall\, 1 \le i \le Number\ of\ \text{Processors} \qquad (2)$$

$$CC(T) = \sum_{i=1}^{number\ of\ new\ processes} \left(r_{c_i f_i} + h_{c_i f_i} \times d_i\right) \qquad (3)$$

Definition 3

The Processor utilization for each processor is obtained by dividing the sum of processing times by maxspan, and the average of processors utilization is obtained by dividing the sum of all utilizations by number of processors (4, 5):

$$U(p_i) = Load(p_i) \Big/ \max span \qquad (4)$$

$$AveU = \left(\sum_{i=1}^{No\ of\ processors} U(p_i)\right) \Big/ Number\ Of\ \text{Processors} \qquad (5)$$

Definition 4

Number of Acceptable Processor Queues (NoAPQ): We must define thresholds for light and heavy load on processors. If the processes completion time of a processor (by adding the current system load and those contributed by the new processes) is within the light and heavy thresholds, this processor queue will be acceptable. If it is above the heavy threshold or below the light-threshold, then it is unacceptable, but what is important is average of number of acceptable processors queues, which is achievable by (6):

$$AveNoAPQ = NoAPQ / Number\ Of\ \text{Processors} \qquad (6)$$

Definition 5

A Queue associated with every processor, shows the processes that processor has to execute. The execution order of processes on each processor is based on queues.

THE PROPOSED GA-BASED ALGORITHM

Genetic algorithms, as powerful and broadly applicable stochastic search and optimization techniques, are the most widely known types of evolutionary computation methods today. In general, a genetic algorithm has five basic components as follows [3]:

1. An encoding method, that is a genetic representation (genotype) of solutions to the program.
2. A way to create an initial population of individuals (chromosomes).
3. An evaluation function, rating solutions in terms of their fitness, and a selection mechanism.
4. The genetic operators (crossover and mutation) that alter the genetic composition of offspring during reproduction.
5. Values for the parameters of genetic algorithm.

Genotype

In the GA-Based algorithms each chromosome corresponds to a solution to the problem. The genetic representation of individuals is called *Genotype*. Many *Genotypes* have been proposed in [3] .In this paper a chromosome consists of an array of *n* digits, where *n* is the number of processes. Indexes show process numbers and a digit can take any one of the *1..m* values, which shows the processor that the process is assigned to. If more than one process is assigned to the same processor, the left to-right order determines their execution order on that processor.

Initial Population

A genetic algorithm starts with a set of individuals called initial population. Most GA-Based algorithms generate initial population randomly. Here, each solution *i* is generated as follows: one of the unscheduled processes is randomly selected, and then assigned to one of the processors. The important point is the processors are selected circularly, it means that they are selected respectively form first to last and then come back to first. This operation is repeated until all of processes have been assigned. An initial population with size of *POPSIZE* is generated by repeating this method.

Fitness Function

As discussed before, the main objective of GA is to find a schedule with optimal cost while load-balancing, processors utilization and cost of communication are considered. We take into account all objectives in following equation. The fitness function of a Schedule T (7):

$$fitness(T) = \frac{(\gamma \times AveU) \times (\theta \times AveNoAPQ)}{(\alpha \times \max span(T)) \times (\beta \times CC(T))} \quad (7)$$

Which $0 < \alpha, \beta, \gamma, \theta \le 1$ are control parameters to control effect of each part according to special cases and their default value is one. This equation shows that a fitter solution (Schedule) has less maxspan, less communication cost, higher processor utilization and higher Average number of acceptable processor queues.

Selection

The selection process used here is based on spinning the roulette wheel, which each chromosome in the population has a slot sized in proportion to its fitness. Each time we require an offspring, a simple spin of the weighted roulette wheel gives a parent chromosome. The probability p_i that a parent T_i is selected is given by (8):

$$P_i = \frac{F(T_i)}{\sum_{j=1}^{POPSIZE} F(T_j)} \quad (8)$$

where $F(T_i)$ is the fitness of chromosome T_i.

Crossover

Crossover is generally used to exchange portions between strings. Several crossover operators are described in the literature [10]. Crossover is not always affected, the invocation of the crossover depends on the probability of the crossover Pc. We have implemented two crossover operators. The GA uses one of them, which is decided randomly.

Single-Point Crossover

This operator randomly selects a point, called *Crossover point*, on the selected chromosomes, then swaps the bottom halves after crossover point, including the gene at the crossover point and generate two new chromosomes called children.

Proposed Crossover

This operator randomly selects points on the selected chromosomes, then for each child non-selected genes are taken from one parent and selected genes from the other.

Mutation

Mutation is used to change the genes in a chromosome. Mutation replaces the value of a gene with a new value from defined domain for that gene. Mutation is not always affected, the invocation of the Mutation depend on the probability of the Mutation *Pm*. We have implemented two mutation operators. The GA uses one of them, which is decided randomly.

First Mutation Operator
This operator randomly selects two points on the selected chromosome, then generates a chromosome by swapping the genes at the selected points.

Second Mutation Operator
The other approach is to check if any jobs could be swapped between processors which would result in a lower make span. If we want to test every possible swap, it would be computationally very intensive, and in larger problems would take an unfeasible amount of time. It also seems unreasonable to consider swapping processes on processors which their load is significantly below the make span, therefore we try to swap processes between overloaded and under loaded processors. This concept can be implemented as follows:

1. First, select a processor, say p_v, which has the latest finish time.
2. Second, select a processor, say p_u, which has least finish time.
3. Third, try to transfer a process form p_v to p_u or swap a single pair of processes between p_v and p_u that improves the make span of both processors the most.
4. This procedure is repeated until no further improvement is possible.

Replacement Strategy

When genetic operators (crossover, mutation) are applied on selected parents T_1, T_2 two new chromosomes *T'* and *T''* are generated. These chromosomes are added to new temporary population. By repeating this operation a new temporary population with size of 2**POPSIZE* is generated. After that fitter chromosomes are selected from current population and new temporary population, at last selected chromosomes made new population and algorithm restarts.

Termination Condition

We can apply multiple choices for termination condition: Max number of generation, algorithm convergence, equal fitness for fittest selected chromosomes in respective iterations.

The Structure of Proposed GA-Based Algorithm

Our proposed GA-Based algorithm starts with a generation of individuals. A certain fitness function is used to evaluate the fitness of each individual. Good individuals survive after selection according to the fitness of individuals. Then the survived individuals reproduce offspring through crossover and mutation operators. This process iterates until termination condition is satisfied. It is Considerable to say that parameters such as $P_C, P_m, POPSIZE, NOGEN, \alpha, \beta, \gamma, \theta$ must be determined before GA is started. Figure 1 shows this operation.

```
Procedure GA-Based algorithm;
Begin
  initialize P(k);  {create an initial population}
  evaluate P(k);  {evaluates all individuals in the
population}
  Repeat
      For i=1 to 2*POPSIZE do
          Select two chromosomes as parent 1
          and parent 2 from population;
          Child 1 and Child 2 ← Crossover( parent1,
          parent2);
          Child 1 ← Mutation ( Child 1 );
          Child 2 ← Mutation ( Child 2 );
          Add (new temporary population, Child 1, Child 2
);
      End For;
      Make (new population, new temporary population,
population );
     Population = new population;
  While (not termination condition);
Select Best chromosome in population as solution and
return it;
End
```

Figure 1: The Structure Of Proposed Algorithm

EXPERIMENTAL RESULTS

In this section, we have used the simulation results to show the performance of the proposed GA-based algorithm. Current solution techniques are concentrated on scheduling tasks with precedence constraints so our approach is not completely comparable with them. We have implemented more than 3000 lines of C++ program to simulate all of the proposed algorithms. All simulation experiments are run on a Pentium III 800, 256 MB RAM, IBM PC. We have tried different values of the population size (*POPSIZE*), mutation Probability (*Pm*),and crossover probability (*Pc*),to find which values would steer the search towards the best solution. The measurement of performance of these algorithms was based on three metrics: total completion time, average processor utilization and, cost of communication. The default parameters were varied and the results collected from test runs were used to study the effects of changing these parameters.

Changing The Number of Processes

We have studied the effect of increasing number of processes on total completion time and average processor utilization. The Obtained results are shown in Figure 2 and, Figure 3. A considerable point in Figure 3 is that when number of processes is increased, higher utilization is obtained. Brief justifications for the values used are given below. When the values discussed were tested 'base values' for each of the parameters where used to help isolate the performance of the parameter in

hand. These values were: *Pc*=0.9, *Pm*=0.1, *POPSIZE*=50, *NOGEN*=50, *m*=10 (number of processors), *n=100...1000*.

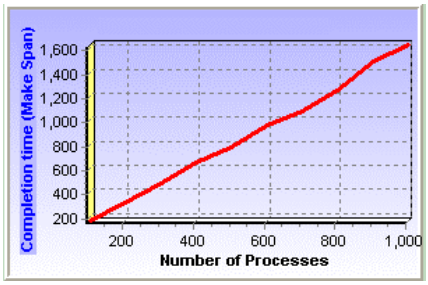


Figure 2: Total Completion Time

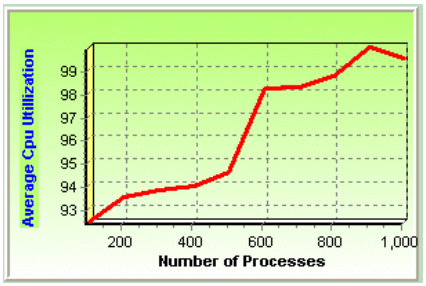


Figure 3: Average Processor Utilization

Changing The Number of Generations

When the number of generations was increased our proposed algorithm had a better function. The Obtained results are shown in Figure 4, Figure 5 and, Figure 6. While the number of generations was increased the total completion time was reduced, it is because the quality of the generated process assignment improves after each generation. A considerable point in these figures is that when the number of generations was increased higher utilization is obtained and, the cost of communication was decreased. Brief justifications for the values used are given below. These values were: *Pc*=0.9, *Pm*=0.1, *POPSIZE*=100, *NOGEN*=50…245, *m*=10 (number of processors), *n=300* (number of processes).

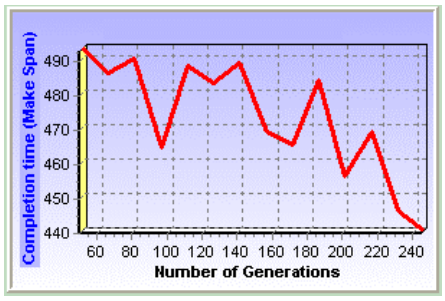


Figure 4: Total Completion Time

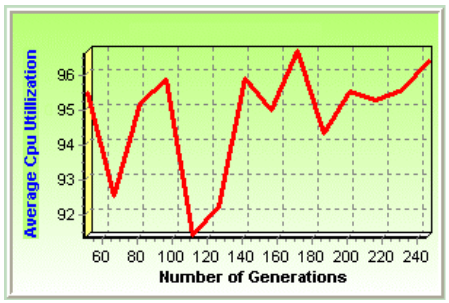


Figure5: Average Processor Utilization

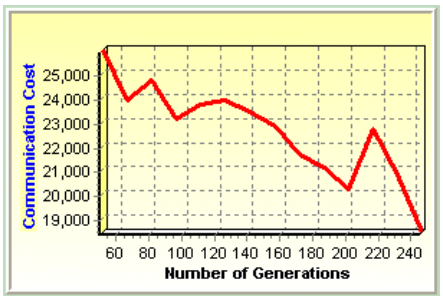


Figure 6: Cost Of Communication

Changing The Size of Population

Changing the size of population is also considerable in terms of total completion time, processor utilization and, cost of communication. The Obtained results are shown in Figure 7, Figure 8 and, Figure 9. While the size of population was increased the total completion time was decreased and, average processor utilization was increased. it is because that the number of the fitter chromosomes which are able to survive was increased, therefore fitter offspring may be generated and it leads to better schedules. According to above results while the size of population was increased higher processor utilization obtained and the cost of communication was decreased. Brief justifications for the values used are given below. *Pc*=0.9, *Pm*=0.1, *POPSIZE*=50…150, *NOGEN*=50, *m*=10 (number of processors), *n=300* (number of processes).

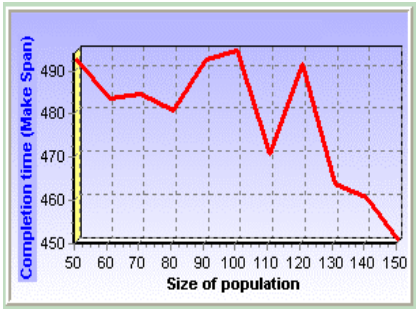


Figure 7: Total Completion Time

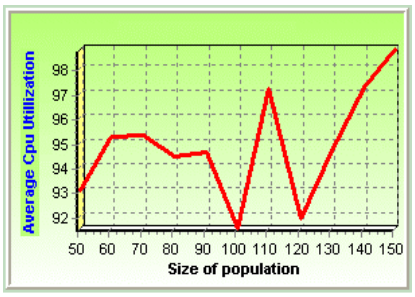


Figure 8: Average Processor Utilization

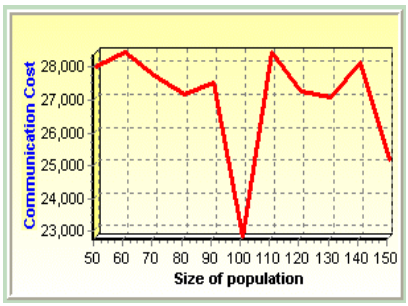


Figure 9: Cost Of Communication

CONCLUTIONS

Scheduling in distributed operating systems has a significant role in overall system performance and throughput. The scheduling in distributed systems is known as an NP-complete problem even in the best conditions. We have presented and evaluated new GA-Based method to solve this problem. This algorithm considers multi objectives in its solution evaluation and solves the scheduling problem in a way that simultaneously minimizes maxspan and communication cost, and maximizes average processor utilization and load-balance. Most existing approaches tend to focus on one of the objectives. Experimental results prove that our proposed algorithm tend to focus on all of the objectives simultaneously and optimize them.

REFERENCES

[1] W.Yao, J.Yao, & B.Li, "Main Sequences Genetic Scheduling For Multiprocessor Systems Using Task Duplication", *International Journal of Microprocessors and Microsystems*, *28*, 2004, 85-94.

[2] G.L.Park, "Performance Evaluation of a List Scheduling Algorithm In Distributed Memory Multiprocessor Systems", *International Journal of Future Generation Computer Systems 20*, 2004, 249-256.

[3] A.T. Haghighat, K. Faez, M. Dehghan, A. Mowlaei, & Y. Ghahremani, "GA-based heuristic algorithms for bandwidth-delay-constrained least-cost multicast routing", *International Journal of Computer Communications 27*, 2004, 111–127.

[4] M. Moore, "An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster", *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2003.

[5] V. D. Martino, "Sub Optimal Scheduling in a Grid using Genetic Algorithms", *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2003.

[6] C.I.Park, & T.Y.Choe, "An optimal scheduling algorithm based on task duplication" , *IEEE Trans. on Computers*, *51*(4), 2002, 444–448.

[7] A.T. Haghighat, K. Faez, M. Dehghan, A. Mowlaei, & Y. Ghahremani, "Multicast routing with multiple constraints in high-speed networks based on genetic algorithms" , *In ICCC 2002 Conf.*, India, 2002, 243–249.

[8] A.Y.Zomaya, & Y.Teh, "Observations on Using Genetic Algorithms for Dynamic Load-Balancing", *IEEE Trans .On Parallel and Distributed Systems, 12*(9), 2001, 899-911.

[9] K.Qureshi, and M.Hatanaka, "A Practical Approach of Task Scheduling and Load Balancing on Hetrogeneous Distributed Raytracing Systems", *Information Processing Letters 79*, 2001, 65-71.

[10] L.M.Schmitt, "Fundamental Study Theory of Genetic Algorithms" , *International Journal of Modelling and Simulation Theoretical Computer Science 259*, 2001, 1 – 61.

[11] A.Y.Zomaya, C.Ward, & B.Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues", *IEEE Trans. On Parallel and Distributed Systems*, 10(8), 1999, 795-812.

[12] S. Salleh, & A.Y. Zomaya, "Scheduling in Parallel Computing Systems: Fuzzy and Annealing Techniques", *Kluwer Academic*, 1999.

[13] M.Lin, & L.T.Yang, "Hybrid Genetic Algorithms for Scheduling Partially Ordered Tasks in a Multi-processor Environment", *Proc. of the 6th IEEE Conf. on Real-Time Computer Systems and Applications*, 1999, 382–387.

[14] Sung-Ho Woo, Sung-Bong Yang, Shin-Dug Kim, Tack-Don Han, "Task scheduling in distributed computing systems with a genetic algorithm", *High-Performance Computing on the Information Superhighway*, HPC-Asia '97, 1997, p. 301.

[15] C. Xu, & F. Lau, "Load-Balancing in Parallel Computers : Theory and Practice", *Kluwer Academic*, 1997.

[16] Y. Lan, & T. Yu, "A Dynamic Central Scheduler Load-Balancing Mechanism", *Proc. of the 14th IEEE Ann. Int'l Phoenix Conf. on Computers and Communication*, 1995, 734-740.

[17] E.S.H.Hou, N.Ansari, & H.Ren, "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Trans. On Parallel and Distributed Systems*, *5*(2), 1994, 113-120.

[18] C.M.Woodside, & G.G.Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", *IEEE Trans. On Parallel and Distributed Systems*, *4*(2), 1993, 164-174.

[19] H.C. Lin, & C.S. Raghavendra, "A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC)", *IEEE Trans. on Software Eng., 18*(2), 1992, 148-158.

[20] A.Nanda, et. al, "Scheduling Directed Task Graphs on Multiprocessors Using Simulated Annealing", *Proc. Int'l. Conf. On Distributed Systems*, 1992, 20-27.

[21] A.K.Sarje & G.Sagar, "Heuristic Model for Task Allocation in Distributed Computer Systems", *Proc. of the IEE-E, 138*(5), 1991, 313-318.

[22] F. Bonomi, & A. Kumar, "Adaptive Optimal Load-Balancing in a Heterogeneous Multiserver System with a Central Job Scheduler", *IEEE Trans. on Computers*, *39*(10) 1990, 1232-1250.

[23] C.C.Shen, & W.H.Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Using a Minimax Criterion", *IEEE Trans. On Computers*, *34*(3), 1985, 197-203.

[24] P.Y.R.Ma, E.Y.S.Lee, & J.Tsuchiya, "A Task Allocation Model for Distributed Computing Systems", *IEEE Trans. On Computers*, *31*(1), 1982, 41-47.

[25] T.C.Hu, *combinatorial algorithms* (Addison-Wesley, 1982).

RESEARCH ON A LOW-POWER MCD TECHNIQUE BASED ON EPIC

Rong Ji, Liang Chen, Yongwen Wang, Xianjun Zeng, Junfeng Zhang
School of Computer Science
National University of Defense Technology
Changsha, Hunan Prov. China, ZIP:410073
E-mail: rongji@nudt.edu.cn

KEYWORDS

EPIC, MCD, GALS, low-power, clock network

ABSTRACT

With the development of very large scale integration (VLSI), the clock frequency of microprocessors rapidly increases, which brings the significant challenge for the microprocessors' power. The multiple clock domain (MCD) technique is a new technique to compromising between synchronous systems and asynchronous systems to reduce the power. Most present studies of MCD are only based on superscalar architectures. In this paper, a MCD technique based on explicitly parallel instruction computing (EPIC) is designed and implemented. Furthermore, a series of experiments on our design have been done to evaluate it. The result of the experiments show that, an EPIC microarchitecture based on MCD technique with a fine-grained adaptive dynamic adjustment algorithm, can effectively decrease the microprocessor power by 40%, compared with the conventional EPIC processor with only one clock domain.

1 INTRODUCTION

With the development of VLSI, the clock frequency of microprocessors rapidly increases, which brings significant challenge for the microprocessors' power. The clock network can dissipate 20–50% of the total power on a chip [1].Thus the optimization of the clock network becomes one of the important issues of high-performance microprocessor design.

At present, most microprocessors are implemented by using fully synchronous mode. The clock distribution network is designed very carefully to meet the constraints of clock skew. It contributes to the complexity of clock interconnection and the significant increase of microprocessor power. So the designers present asynchronous system which need not clock. But there are so many difficulties in design of the complete asynchronous signals. Globally Asynchronous Locally Synchronous (GALS) [2], which is a compromise between asynchronous systems and synchronous systems, has been focused on.

Almost current researches on GALS are based on superscalars [3,4,5,6,7,8], and not been applied to EPIC architecture, because there are much difficulties in its implementation. For example, the uncertainty of asynchronous communication brings significant challenges for data dependence and instruction completion in order in EPIC.

This paper describes the research on MCD based on EPIC is done with GALS style, which includes the implementation of it, and experimental simulation to evaluate it. Section 2 presents the conventional EPIC architecture, and analyzes what limits power optimization of microprocessor. Section 3 describes the design and implementation of the MCD based on EPIC. Section 4 details the simulation framework and experimental setup. The results of all the tests are given and analyzed in Section 5. Conclusions and future work are in Section 6.

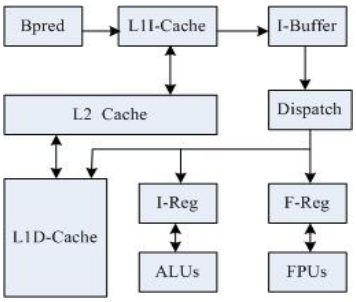
2 THE CONVENTIONAL EPIC ARCHITECTURE

2.1 The Conventional EPIC with One Clock Domain

An EPIC architecture emphasizes co-operating compiler with hardware to exploit instruction-level parallelism (ILP). It combines the advantages of superscalar architectures and the advantages of very long instruction word (VLIW) architectures [9]. Itanium 2 [10] is a classical processor based on EPIC architecture. Thus the EPIC architecture of Itanium 2 processor is a comparable architecture to our design.

Figures 1 shows a conventional EPIC architecture with single clock domain. The front-end of the microprocessor obtains the addresses of fetching instructions, by means of Bpred predicting branch objects. The instructions are fetched from instruction Cache (I-Cache), and reserved into instruction buffers (I-Buffer). They are decoded by the dispatch logic (Dispatch), and sent to corresponding execution units. ALUs and FPUs can execute multiply independent integer instructions and floating-point instructions per cycle, respectively. The operands are respectively given by integer registers (I-Reg) and floating registers (F-Reg). The results are reserved in corresponding registers. Load/store operations will access Level 1 data Cache (D-Cache) and level 2 Cache (L2 Cache). There are abundance of functional units and Cache levels to provide adequate hardware supports for exploiting ILP.

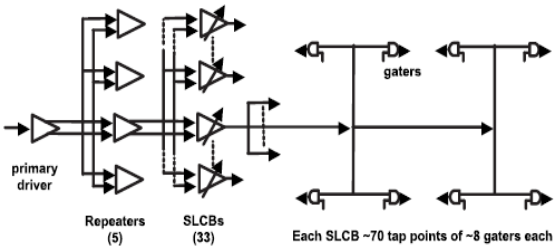
Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)



Figures 1: A Conventional EPIC Architecture

2.2 The Analysis What Limits Power Optimization of the Conventional EPIC Microprocessors

Currently the microprocessors based on the conventional EPIC architecture often use the globally synchronous clock strategy, such as the clock distribution network of the Itanium 2 processor described as Figures 2. The clock signal is generated by a clock generator, and is sent to each unit within the microprocessor by the long wires and a lot of buffers. All units synchronously operate under control of this global clock signal. The clock distribution network is usually designed carefully to meet the constraints of clock skew. It contributes to the complexity of clock interconnection and the significant increase of microprocessor power. Thus, it is necessary for EPIC architecture to implement MCD with GALS style, in order to decrease the microprocessor power.



Figures 2: The Clock Distribution of Itanium 2

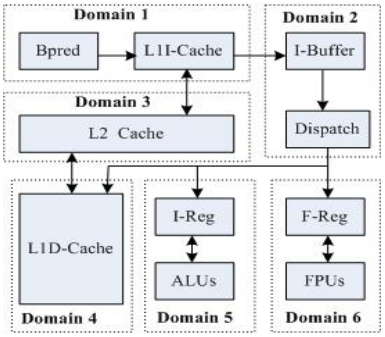
3 THE DESIGN AND IMPLEMENTATION OF MCD BASED ON EPIC

A MCD architecture based on EPIC, which need no global clock, is called MCDE (multiple clock domain based on EPIC). The MCDE microprocessor can be divided into several clock domains. Each domain uses an independent clock. Among different domains, there is asynchronous communication which can effectively save the power of microprocessors.

3.1 The Partition of Clock Domains

In MCDE, the entire chip can be divided into several domains according to the EPIC architecture characteristic of the microprocessor. There are two rules in the partition: 1) this partition can't change the organization structure of the microprocessor pipelines too much; 2) the domain boundaries are set between the components having a loose coupling with each other to the best of our abilities. The components having a close coupling with each other are placed in the same domain to reduce communication operations among different domains

The EPIC microprocessor is partitioned to six clock domains according to the rules described above: a fetching instruction domain (Domain1), a dispatch domain (Domain2), a L2 Cache domain (Domain 3), a load/store domain (Domain4), an integer domain (Domain5), and a floating-point domain (Domain6). Figures 3 shows the partition detail. The function of Domain1 contains branch prediction, instruction address generation, and I-Cache read. Domain2 accomplishes dispatch of instructions. Domain3 includes L2 Cache read/write operation. L1D-Cache read/write is completed in Domain4. Domain5 completes the load/store operation of integer operands, and execution of arithmetic logic. Domain6 consists of the load/store operation of floating-point operands, and execution of floating-point computing. In our design, Domain1 and Domain2 are called front-end. Domain3, Domain4, Domain5, and Domain6 are called back-end. Each domain has its own local clock. The units within same domain operate in synchronous mode. The asynchronous communication is used among different domains.

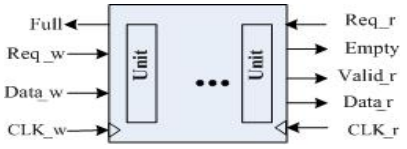


Figures 3: The Clock Domain Partition of EPIC

3.2 Communication Mechanisms among Domains

Within the microprocessor, the queue structures are synchronous points of the different domains. In this case, when the queues are not full and not empty, the latencies of synchronous overhead can be hidden. The design of the queue structures is based on the hybrid clock FIFO, which is shown as Figures 4. The FIFO queue has two ports: input and output. The function of input is to write data to the FIFO, which is controlled by *CLK_w*. *Req_w* is the control signal of write requirement. *Data_w* is the input data bus. When the FIFO is full, the signal *full* is set one. The function of the output of the FIFO is to read data from the FIFO, which is controlled by *CLK_r*. *Req_r* is the control

signal of read requirement. *Data_r* is the output data bus. *Valid_r* is the valid bit of output data.



Figures 4: The Hybrid Clock FIFO

The process of data stored at input is done as follows: When the rising edge of *CLK_w* arrives, *Req_w* that is the requirement signal and the data in *Data_w* are sampled. The data items will be written into the queue at the next clock period. If the *FIFO* is full, the signal *Full* is set before the next clock period arrives, and receiving the data at the input is prohibited.

The process of reading data from the output is described as follows: When the rising edge of *CLK_r* arrives, *Req_r* that is the requirement signal is sampled. The data items will be put on *Data_r* bus, and the valid bit *Valid_r* is set. If the *FIFO* is empty at this clock period, the signal *Empty* is set, and reading operation at the output is prohibited until the *FIFO* is not empty.

3.3 A Dynamic, Adaptive Control Algorithm of Clock Domain's Frequency

By analyzing processor resource utilization, a correlation is revealed, over an interval of instructions, between the valid entries in the input queue and the desired frequency for the domain. Queue utilization is thus an appropriate metric for dynamically determining the desired domain frequency. The dynamic, adaptive control algorithm of clock domain's frequency is based on this idea to reduce the power consumption of the clock network.

The dynamic, adaptive control algorithm of clock domain's frequency is described as follows: The MCDE architecture uses the attack/decay algorithm independently in each back-end domain. When the entries in the domain issue queue is in excess of 10,000-instruction interval, the hardware counts. Using the number and the corresponding number from the previous interval, the algorithm determines whether there is a significant change that threshold is 1.7 percent, in which case the algorithm uses the attack mode: The frequency changes by 7 percent. If no significant change occurs, the algorithm uses the decay mode: It reduces the domain frequency slightly by 0.17 percent.

But when the instructions per cycle (IPC) changes by more than a certain threshold that is 2.3 percent, the frequency remains unchanged for that interval. This rule is used to identify natural decreases in performance that are unrelated to the domain frequency, and to prohibit the algorithm from reacting to them.

4 THE SIMULATION FRAMEWORK AND EXPERIMENTAL SETUP

4.1 The Power Consumption Evaluation Model

To evaluate our MCDE architecture, we use basic technology parameter of CACTI 0.8μm, and scaling down method to implement power consumption evaluation model [11].The delay components that make up a general system are composed of the following three individual subsystems:

- **Memory Storage Elements**: Those elements include the Cache, the registers, all kinds of queue, and buffers in the processor. It is divided into five parts: the address decoders (*dec*), the bit lines (*bl*), the word lines (*wl*), the sense amplifiers (*amp*), and the route components (*rt*) of address and data bus. Thus, P_{Mem}, the peak power of memory is the sum of the maximum power of these five parts:

$$P_{Mem} = P_{dec} + P_{bl} + P_{wl} + P_{amp} + P_{rt}$$

Assume that, C represents the capacitance of each kinds of memory; V and f denote the working voltage and frequency, respectively; V_{dd} is the voltage supply; the maximum toggle rate of the decoder signal is 0.3, that of other parts is 1.

The power of the address decoder P_{dec} is given by

$$P_{dec} = 0.3C_{dec}V^2 f$$

The power of the word line P_{wl} is given by

$$P_{wl} = C_{wl}V^2 f$$

The power of the bit line P_{bl} is given by

$$P_{bl} = C_{bl}V^2 f$$

The power of the sense amplifier P_{amp} is computed by the empirical formula.

$$P_{amp} = 0.5C\frac{V_{dd}}{8}\times 10^{-3}$$

The power of the route component P_{rt} is given by

$$P_{rt} = C_{rt}V^2 f$$

- **Logic elements**: Those elements include such units as ALU and floating-point unit. For the power evaluation of the logic elements, we first use the Wattch power model [12], and the capacitance value of the basic device and wire under technology parameter of CACTI 0.8μm [13] to compute the reference power P_P, on condition that reference voltage is V_P and reference frequency is f_P. Assume that, the characteristic

size of the actual technology of the microprocessor is f_s. According to the scaling down theory of CMOS [14], for the given voltage and frequency, the power of the logic element is computed by

$$P_{logic}(f_s, V, f) = \frac{V^2 f}{\frac{0.8}{f_s} V_P^2 f_P} P_P$$

- **Clock network**: Assume that, the microprocessor is divided into M domains; For the i*th* domain, the capacitance of the local clock wire is $C_w(CL_i)$, the load capacitance of the local clock is $C_t(Cp_i)$, the load capacitance of the local clock buffer is $C_t(B_i)$, the load capacitance among the local pipelines is $C_t(P_i)$, the working voltage and frequency is V_i and f_i, respectively. The peak power of the clock network is given by

$$P_{mclk}(f_s, V, f, M) = \sum_{i=1}^{M} (C_w(CL_i) + C_t(Cp_i) + C_t(B_i) + C_t(P_i)) V_i^2 f_i$$

If M = 1, it represents the peak power of the clock network with single clock domain.

After the peak power of each part is evaluated, the performance simulator is run to obtain dynamically the statistic of the access counts of each component per cycle. The dynamic power is computed according to the statistic and the peak power of each part.

4.2 The Simulation and Experimental Setup

Since IMPACT provides a comprehensive infrastructure for modeling and simulation of EPIC microarchitecture feature, the Lsim simulator [15] of the IMPACT compile framework is adopted as the simulation engine. Furthermore, to simulate the MCDE processor, the event-driven algorithm based on multi-clock domains is designed, described as Algorithm 1.

```
Init_event_queue();                    /*  initialize the event queue  */
while ((event_queue is not empty) && (simulation is not finished))
  Get the id number(id), cycle time(cycle),
                          and trigger_time(trigger_time);
  timer = trigger_time;    /*  push ahead with the global timer  */
  event_handler();         /*  call the the handler of this event  */
  next_time = timer +cycle;
                  /*  arrange next occurrence time of this event  */
  Attain the tail pointer pt of the current queue;
  while( (next_time < the occurrence time pressed toward by pt)
        || (the priority of the current event is more prior))
    pt = pt→prev_event;    /*  move the pointer ahead  */
  if (pt is empty)
     Take error alarm: the event occurrence time is passed;
  else
     Insert the current_event_node subsequent to the node
                                  pressed toward by pt.
```

Algorithm 1: The Event-driven Simulation Algorithm

To evaluate objectively the MCDE proposed, we design three groups of experiments:

- **SVF Strategy**: Each of clock domains works on condition that the voltage is same, and frequency is also same, called SVF for simple,which represents same voltage and frequency. That is to verify the performance and power consumption characteristic of MCDE relative to the EPIC.

- **DVF Strategy**: Each of clock domains works on condition that the voltage may be different, and frequency may be different, too. The frequency and voltage are not adjusted during the system running. This strategy is called DVF for simple, which denotes different voltage and frequency. It can verify advantages of fine-grained setting of voltage and frequency in MCDE. In this group of experiment, the voltage and the frequency of each of domains are set as Table 1, according to the architecture characteristic of Itanium 2.

Table 1: Different Voltage and Frequency Setting

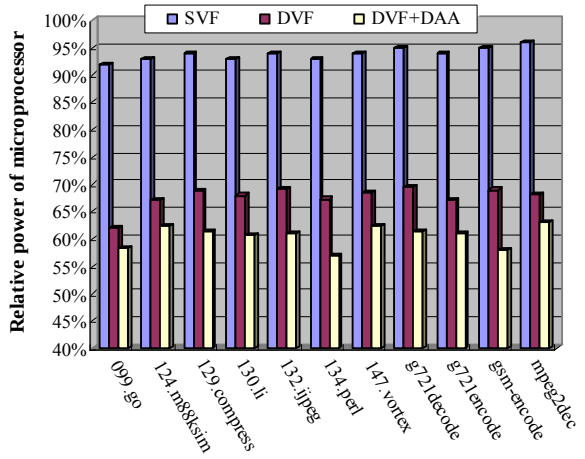
Domain	Frequency (MHz)	Voltage (V)
Domain1	1000	1.5
Domain2	1000	1.5
Domain3	100	0.8
Domain4	500	1.1
Domain5	500	1.1
Domain6	0	0

- **DVF+DAA Strategy**: Each clock domains works on condition that voltage and frequency maybe different with each other. In this group of experiment, the voltage and the frequency of each domain is also set as Table 1. Furthermore, during the system running, the frequency of each back-

end domains is adjusted dynamically and adaptively as the control algorithm described above. This strategy is called DVF+DAA, which is an abbreviation for Different Voltage and Frequency + Dynamic Adaptive Adjustment Strategy. This group of experiment is used to verify the potentiality for fine-grained dynamic adaptive frequency adjustment decreasing the power of the microprocessor in MCDE.

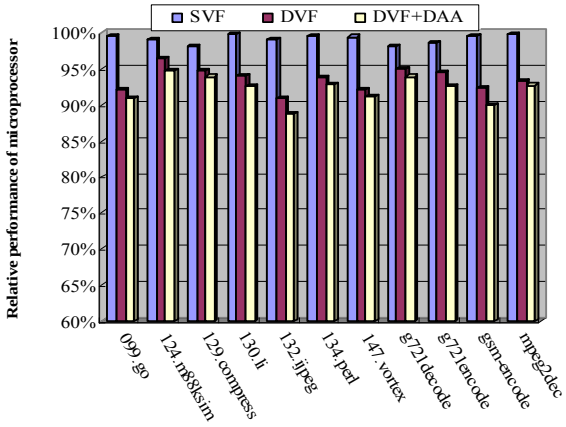
5 THE ANALYSIS OF EXPERIMENT RESULTS

Figures 5 shows the power consumption of MCDE processors relative to the basic EPIC processor under the three groups of experiment described above. The results of the experiments reveal that all three MCDE strategies can decrease the microprocessor's power consumption. But the power decrease of SVF is small, which is only 6 percent. After DVF used, the processor's power is decreased by 32 percent owing to further exploiting the advantage of MCDE. Comparing with SVF and DVF, Using DVF+DAA strategy can more effectively decrease the power consumption of the microprocessor, which decreases the power by 40 percent, as a result of using the fine-grained dynamic adaptive frequency adjustment.



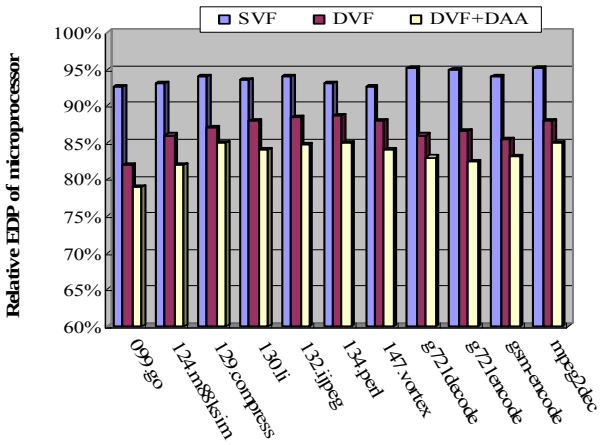
Figures 5: The Power of MCDE Relative to EPIC

Figures 6 shows that using the different MCDE strategy has an impact on the performance of the micro-processor. The results of the experiment reveal that all these three MCDE strategies can lead to performance degradation relative to EPIC, since they use the asynchronous communication mechanism. The SVF strategy results in a slight degradation, within 1 percent. Comparing with SVF, DVF and DVF+DAA result in more performance degradation owing to clock frequency being decreased. For DVF, the average performance degradation is approximately 6.5 percent. For DVF+DAA, the average performance degradation is about 7.3 percent.



Figures 6: The Performance of MCDE Relative to EPIC

When designing architecture oriented towards power consumption, it is improper that only performance or only power consumption is considered. We should comprehensively consider the performance and the power consumption, and compromise between them. Energy delay product (EDP) is a popular metric to evaluate comprehensively the performance and the power consumption. Figures 7 shows the EDP of MCDE relative to EPIC, corresponding to three MCDE strategies. The results of the experiment indicate that, for DVF+DAA, although the performance degradation is slightly significant, a significant overall EDP improvement is achieved, about 17 percent, owing to immensely exploit the potential for MCDE reducing the power consumption of the microprocessor. In addition, for DVF, the average EDP improvement is approximat-ely 14 percent. Finally, for SVF, although the performance degradation is very slight, comparing with DVF and DVF+DAA, the power consumption optimization brought by SVF is not significant, only 7 percent, as a consequence of the conservative strategy. But comparing with EPIC, all three MCDE strategies are obvious to improve the EDP. Thus, there is a significant advantage of decreasing the power consumption by using MCDE, comparing with EPIC.



Figures 7: The EDP of MCDE Relative to EPIC

6 CONCLUSIONS AND FUTURE WORK

The MCD is a novel technique that combines the advantages of synchronous systems and the advantages of asynchronous systems to resolve the problems of high-performance microprocessors, such as the significant power, and the complex clock distribution network. At present, the studies of MCD are almost based on superscalar. In this paper, MCDE, a MCD technique based on EPIC, is implemented, and evaluated comprehensively. The experimental results show that, the MCDE has a significant potential for reducing the power consumption of the high-performance EPIC microprocessor. For example, using the DVF+DAA strategy, the power consumption of the EPIC microprocessor can be reduced by 40%, and about a 17 percent EDP improvement is achieved.

In future work, we will continue to develop more effective fine-grained dynamic, adaptive voltage and frequency adjustment mechanisms to attain more significant power consumption decrease at the cost of slight performance degradation.

ACKNOWLEDGMENT

This work was supported by NSFC (No.60676016) in China.

REFERENCES

[1] Jatuchai Pangjun et al.. 2002. "Low-Power Clock Distribution Using Multiple Voltages and Reduced Swings." *IEEE Trans.on VLSI ststems*, Vol.10, No.3(June), 309-318.
[2] D.M. Chapiro. 1984. "Globally Asynchronous Locally Synchronous Systems". PhD thesis, Stanford Univ.
[3] A. Iyer and D. Marculescu. 2002. "Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors". *Proc. 29th Int'l Symp. Computer Architecture* (ISCA 02).IEEE, 158-170.
[4] G. Magklis et al.. 2003. "Profile-based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor". *Proc. 30th Int'l Symp. Computer Architecture* (ISCA 03), ACM Press, 14-27.
[5] G. Semeraro et al.. 2002. "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling". *Proc. 8th Int'l Symp. High-Performance Computer Architecture* (HPCA02), IEEE, 29-40.
[6] G. Semeraro et al.. 2002. "Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture". *Proc. 35th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO-35), IEEE, 356-370.
[7] Iyer A and Marculescu D. 2002. "Power efficiency of voltage scaling in multiple clock, multiple voltage cores". In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD 2002), San Jose, America, 379-386.
[8] Eswaran A and Chen S. 2003. "All-domain fine grain dynamic speed/voltage scaling for GALS processors". URL http://www.ece.cmu.edu/~schen1/ece743/proposal_ up.pdf.
[9] Schlansker M.,Rau B..2000."EPIC: Explicitly Parallel Instruction Computing". *IEEE Computer*, Vol.33, No.2,37-45.
[10] Naffziger S., Colon-Bonet G., Fischer T. et al.. 2002. "The Implementation of the Itanium 2 Microprocessor". *IEEE Journal of Solid-State Circuits*, Vol.37, No.11, 1448-1460.
[11] Wang Yongwen and Zhang Minxuan. 2004. "Microarchitecture-Level Power Modeling and Analyzing for High-Performance Microprocessors". *Chinese Journal of Computers*, Vol.27, No.10(Oct.), 1320-1327.
[12] Brooks D., Tiwari V., Martonosi M.. 2000. "Wattch: A Framework for Architecture-level Power Analysis and Optimizations". *In Proceedings of the 27th Annual International Symposium on Computer Architecture.* Vancouver, British, Columbia, Canada, 83-94.
[13] Reinman G. and Jouppi N.. 2000. "An Integrated Cache Timing and Power Model". Technical report 2000/7, Western Research Laboratory, Palo Alto, California, USA.
[14] Gan Xue-Wen. 1999. "Digital CMOS VLSI Analysis and Design Basics". Peking University Press. Beijing, Chnia.
[15] Chang P., Mahlke S., Chen W. et al.. 1991. "An Architectural Framework for Multiple-instruction-issue Processors". *In Proceedings of the 18th Annual International Symposium on Computer Architecture*, Toronto, Ontario,Canada, 266-275.

AUTHOR BIOGRAPHIES

RONG JI received the M.S. degree in computer engineering from National University of Defense Technology, Changsha, China, in 2004, and is currently working toward the Ph.D. degree at this school. Her research interests include architecture,low-power design, and application analysis.
Her e-mail address is : rongji@nudt.edu.cn

LIANG CHEN received the M.S. degree in computer engineering from National University of Defense Technology, Changsha, China, in 2005, and is currently working toward the Ph.D. degree at this school. His research interests include architecture and low-power design.
His e-mail address is : liangchen@nudt.edu.cn

YONGWEN WANG is a associate professor in School of Computer Science at the National University of Defense Technology. His research interests include computer architecture, and low-power design. He has a Ph.D degree in computer engineering from National University of Defense Technology.
His e-mail address is : ywwang@nudt.edu.cn

XIANJUN ZENG is a professor in School of Computer Science at the National University of Defense Technology. His research interests include computer architecture, and low-power design. He has a Ph.D degree in electrical and computer engineering from National University of Defense Technology.
His e-mail address is : xjzeng@nudt.edu.cn

JUNFENG ZHANG received the M.S. degree in computer engineering from National University of Defense Technology, Changsha, China, in 2006, and is currently working toward the Ph.D. degree at this school. His research interests include low-power design, and application analysis.
His e-mail address is : jfzhang@nudt.edu.cn

SIMULATION BASED OPTIMIZATION OF INDIRECT ALUMINUM EXTRUSION PROCESS PARAMETERS

Sachin Man Bajimaya, Chang Mok Park and Gi-Nam Wang
Ajou University
Department of Industrial & Information Systems Engineering
Suwon, 443-749, South Korea
E-mail: {sachin, cmpark, gnwang}@ajou.ac.kr

KEYWORDS
Extrusion, Simulation, Optimization, IGRIP

ABSTRACT

The objective of this paper is to optimize the principal extrusion variables by means of manufacturing simulation of the extrusion model. The finite element analysis of the extrusion model does not consider the manufacturing process in its modeling. Therefore, the process parameters obtained through such an analysis remains highly theoretical in that these parameters differ from the actual extrusion process parameters. Additionally, due to the nature of the extrusion process, it is often quite difficult to determine the cause of an extrusion problem and find its proper solution, particularly if it must be done quickly. The manufacturing simulation of the extrusion process prior to plant execution helps to make the actual extrusion operation more efficient because more realistic parameters may be obtained through the simulation. Additionally, troubleshooting the process becomes simpler. In this paper, we have developed the simulation model of a real extrusion plant using IGRIP. The model of the plant is simulated in IGRIP using the Graphical Simulation Language (GSL). The dynamics of the plant is also incorporated in the simulation model. Through simulation, optimized values of variables that affect the extrusion process are obtained**.**

INTRODUCTION

One of the greatest challenges in an actual extrusion operation is efficient and rapid problem solving. However, the conventional finite element analysis does not consider the manufacturing process constraints in its modeling and hence, the process parameters obtained through such an analysis is more theoretical and not realistic enough. Therefore, a new approach to obtain more realistic and optimized process parameters following the finite element analysis is the need of manufacturing industries.

Extrusion problems often result in downtime and/or out-of-specification products, and this can be very costly. The nature of the extrusion process is often complex. Therefore, the determination of an appropriate solution to the cause(s) of any problem in the extrusion process is difficult, particularly if the solution must be found quickly. Linear programming and other iterative methods have been used to tackle the extrusion process problems. However, due to the inherent time consuming nature of such methods, quick and rapid problem solving as desired by industries has not been achievable. Additionally, today, the process development in industrial extrusion is to a great extent based on trial and error and often involves full-size experiments, which are both expensive and time-consuming. In contrast, simulation of the extrusion process prior to plant execution and as and when required, proves to solve the extrusion operation problems quickly and more efficiently and at a lower cost. In this paper, model of a real extrusion plant is built in IGRIP, a Delmia group software from Dassault Systems [12]. The model of the plant is simulated in IGRIP using the Graphical Simulation Language (GSL) whereby the virtual operation of the plant may be viewed. Additionally, the dynamics module offered by Delmia is used to model and simulate the extrusion plant dynamics. Simulation is performed to obtain the optimized values of the principal variables that affect the extrusion process.

BACKGROUND

Sivaprasad, Venugopal, Davies and Prasad [7] have identified the optimum process parameters using finite element simulation. They discuss the use of processing map with the output of the finite element analysis to design the process. Tibbetts and Ting-Yung [8] have used optimization technique for a direct extrusion machine. Their work is related to product optimization with focus on surface quality and micro-structural uniformity of product. They have presented a model which is derived directly from the mathematical description of the physical phenomena present. Hansson, in her Ph.D. thesis [9], has used finite elements method for the simulation of stainless steel tube extrusion.

Most other extrusion process simulations have been done for food industries like by Lertsiriyothin and Kumtib [10], and plastic or polymer manufacturing units such as by Salazar [11].

Plenty of studies related to finite element-based models and mathematical descriptions as mentioned above have already been carried out. The finite element models provide the needed information but cannot be applied for direct and fast decision making. The finite element

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

analysis, in particular, does not consider the manufacturing process in its modeling and so, the process parameters obtained through such an analysis differs form the actual manufacturing process parameters. In such a scenario, following the finite element analysis, optimization of process parameters based on the manufacturing simulation of the process yields a direct solution for the real process in that the appropriate process parameters can be recognized directly. The indirect extrusion simulation for the optimization of process parameters dealt with in this paper is thus found to be new and the simulation-based approach is found to be novel in the area. Such an approach is found to provide quick and efficient solution to extrusion problems.

INDIRECT EXTRUSION

The extrusion process

Extrusion is a plastic deformation process in which a block of metal, called the billet, is forced to flow by compression through the die opening of a smaller cross-sectional area than that of the original billet [4] as shown in Fig. 1. In indirect extrusion process, the die at the front end of the hollow stem moves relative to the container, but there is no relative displacement between the billet and the container as depicted in Fig. 1. Therefore, this process is characterized by the absence of friction between the billet surface and the container, and there is no displacement of the billet center relative to the peripheral regions.

Extrusion can be cold or hot. In this paper, we consider the hot extrusion process. In hot extrusion, the billet is preheated to a certain temperature to facilitate plastic deformation.

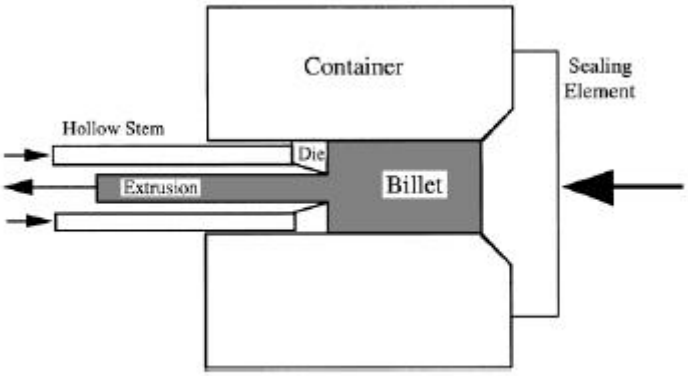


Figure 1: Indirect extrusion mechanism

The properties of the extruded aluminium shapes are affected greatly by the way in which the metal flows during extrusion. The metal flow is influenced by several factors of which the following are considered in this paper:

1. the size of billet (length and diameter)
2. the temperature of billet, $\theta_{preheat}$
3. the temperature of container
4. the extrusion ratio, ER
5. speed of extrusion, $V_{extrusion}$
6. the extrusion pressure, $P_{T(ER)}$

Interdependence between extrusion variables

During the operation of an extrusion plant, while extrusion is taking place, billets will be waiting for their turn to get loaded to the container for extrusion. In hot extrusion, the billets are preheated. It should be assured that the next billet in queue is heated to the required preheat temperature ($\theta_{preheat}$) during the time (τ_{wait}) it waits in the queue. For efficient heat usage, τ_{wait} would be approximately equal to the sum of the time taken by the current billet to be extruded ($\tau_{extrusion}$) and the time taken for the change of die ($\tau_{dieChange}$). Therefore, considering the time taken to preheat to be $\tau_{preheat}$,

$$\tau_{preheat} = \tau_{wait} = \tau_{extrusion} + \tau_{dieChange}$$

Temperature is one of the most important parameters in extrusion. The flow stress (σ) is reduced if the temperature is increased and deformation is therefore, easier, but at the same time, the maximum extrusion speed is reduced because localized temperature can lead to incipient melting temperature.

The response of a metal to extrusion processes can be influenced by the speed of deformation. Increasing the ram speed produces an increase in the extrusion pressure. The temperature developed in extrusion increases with increasing ram speed.

Thus, it is important to determine the optimal values of the principal extrusion variables for a specified extrusion ratio (ER) such that, while on the one hand, the aluminum billet does not reach its solidus point ie, it's melting temperature, and on the other hand, efficient extrusion is assured. Extrusion ratio (ER) is the ratio of container bore area to the total cross-sectional area of extrusion. For an extrusion process, ER is fixed.

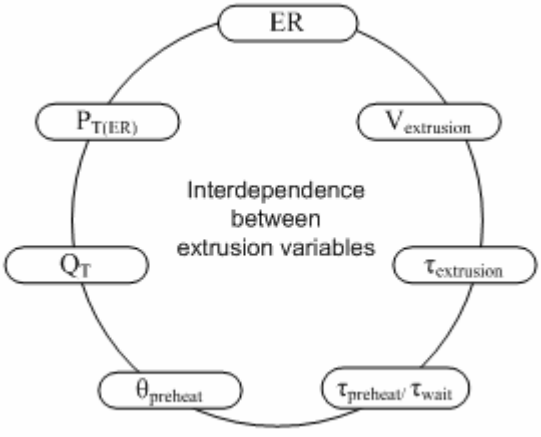


Figure 2: Interdependence between extrusion variables

Determination of ram speed ($V_{extrusion}$) is the most important for a particular extrusion ratio. The ram speed affects the extrusion time ($\tau_{extrusion}$). The preheat time

($\tau_{preheat}$), in turn, is related to the extrusion time. Therefore, simulation is performed for a specific extrusion ratio and speed of ram to obtain the extrusion time. The extrusion time is used to decide the amount of heat (Q_T) that a waiting aluminum billet should be given to reach the preheat temperature ($\theta_{preheat}$) during the period of its waiting time (τ_{wait}).

The actual pressure exerted on the ram is the total pressure. The total extrusion pressure required for a particular extrusion ratio (ER) is given by:

$P_{T(ER)} = P_D + P_F + P_R$

where, P_D is the pressure required for the plastic deformation of the material. P_F is the pressure required to overcome the surface friction at the container wall friction, dead metal zone friction, and die bearing friction. P_R is the pressure required to overcome redundant or internal deformation work.

Each of the above may be represented in functional form as:

P_D = **f** (flow stress σ, strain ε)

where, the flow stress σ = **f** (strain ε, strain rate, temperature of material T).

Here, strain $\varepsilon = \ln(A_C/A_E)$ where, A_C = area of container and A_E = area of extrusion

P_F = **f** (billet diameter D, length of billet L)

P_R = **f** (flow stress σ)

The extrusion pressure, $P_{T(ER)}$, is thus dependent upon the size of the billet, the extrusion ratio, the temperatures of billet and container, flow stress and the strain rate of aluminum. The pressure, $P_{T(ER)}$, so obtained is used to find the extrusion speed ($V_{extrusion}$). Generally, a chart is provided by the extrusion machine manufacturer that gives the extrusion speed required to produce a particular extrusion pressure P_T at a specified temperature and extrusion ratio (Fig. 3, 4 and 5). The extrusion speed may be utilized to find the extrusion time ($\tau_{extrusion}$). The extrusion time, in turn, may be used to find the time required by an aluminum billet to wait (τ_{wait}) before it gets loaded to the container for extrusion. This amount of time is used to find the amount of heat (Q_T) that should be supplied to the aluminum billet during this time (τ_{wait}) to reach its preheat temperature ($\theta_{preheat}$). The values of $\theta_{preheat}$ and $V_{extrusion}$ should be such that the aluminum billet does not reach its solidus temperature, which is approximately 660ºC [3], during the actual extrusion process; which is to say, after the $V_{extrusion}$ is known, $\theta_{preheat}$ should be so selected as to ensure that the aluminum billet does not reach its solidus temperature during extrusion.

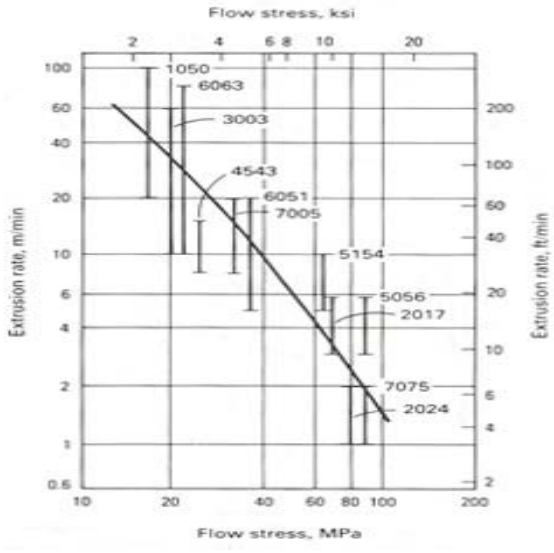


Figure 3: Extrusion rate versus flow stress

The values of the principal variables so obtained may be valid only for theoretical use. In reality, the values of these variables may differ for a practical extrusion process. Also, these values may not be within the permissible values of a particular extrusion machine. So, a simulation is performed, considering the dynamics of the process – stress and strain factors, changes in temperature and pressure, the friction and speed of extrusion – and the permissible value of process parameters to validate the process with the values so obtained.

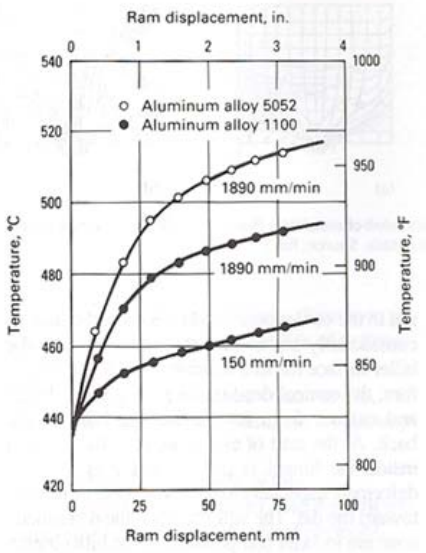


Figure 4: Surface temperature of extruded product versus ram displacement

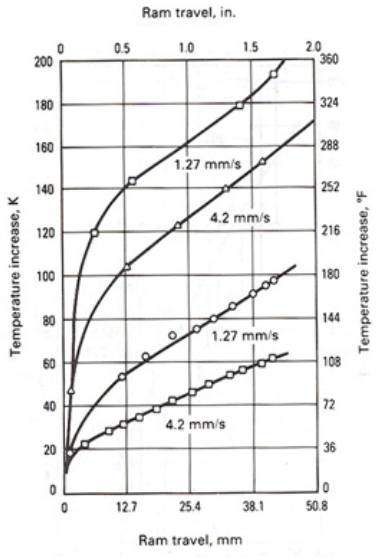


Figure 5: Increase in emergent temperature versus ram speed

IGRIP offers a dynamics module that facilitates the incorporation of the dynamics of a process. Any necessary changes may then be made and simulation may be performed several times until the values of the variables satisfy usage in a real extrusion process. For example, if after calculations, the billet temperature obtained will lead to the aluminum billet reaching its solidus temperature, then the extrusion speed, $V_{extrusion}$, may be reduced. The reduced extrusion speed may be used to find a different extrusion pressure, $P_{T(ER)}$ and also to calculate the extrusion time, $\tau_{extrusion}$. This extrusion time, in turn, may be used to find the time for the billet to wait, τ_{wait} before it gets loaded to the container for extrusion. Also, the amount of heat, Q_T that has to be supplied to the aluminum billet during τ_{wait} may be calculated.

THE EXTRUSION SIMULATION FRAMEWORK AND IMPLEMENTATION

A graphical model of the extrusion plant is built using IGRIP (Fig. 6). The model is validated against the working of a real machine. The validated model is then simulated using the Graphical Simulation Language (GSL) in IGRIP. The dynamic properties of the extrusion process are taken into account in the extrusion model by making use of the dynamics module in IGRIP.

A client software was built using Visual C++. Values of the principal variables for extrusion are sent as inputs from this client (Fig. 7). The server makes use of these principal variables for the simulation of the extrusion model. Additionally, we developed an interface program between the client and the server so that message sent from the VC++ client is read into the GSL program in the server. The GSL program makes necessary readings from

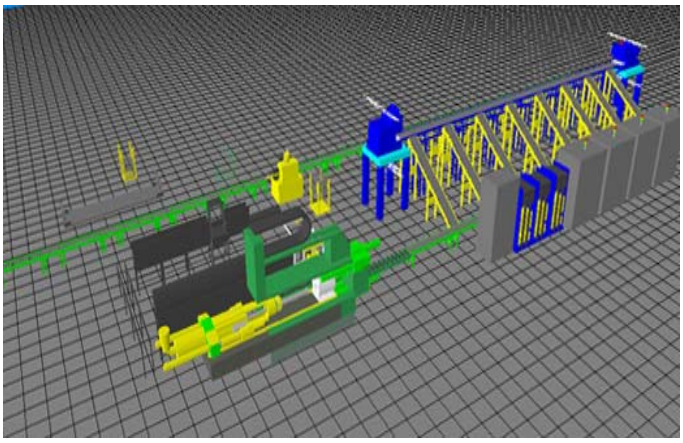


Figure 6: The IGRIP model of the extrusion plant used for simulation

and writings to a database that contains the permissible values of extrusion variables and tables containing the values of extrusion speeds required to produce corresponding extrusion pressures. The database also contains further vendor-specific data regarding extrusion.

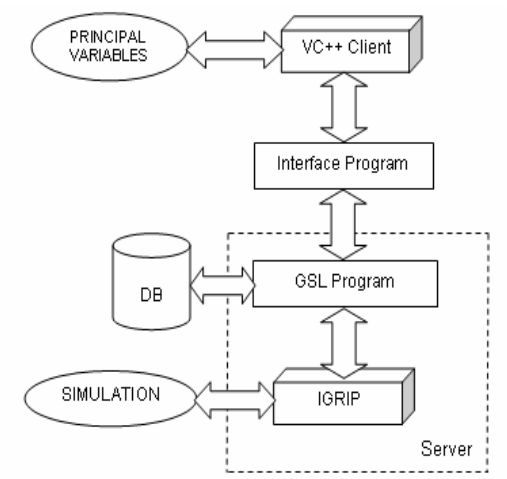


Figure 7: The implementation architecture

The use of the optimized values of the principal extrusion variables in several simulations of the plant resulted in a reduced cycle time of the plant. The minimum decrease in cycle time was 2 minutes which makes a decrease of 8.33 percent whereas the maximum decrease was of 12 minutes that accounts for a 37.5 percent decrease. The average decrease was approximately 5 minutes which is about 20.8 percent (Fig. 8). The power consumed by the complete extrusion operation was also decreased considerably in the simulations of extrusions at varying extrusion ratios (ER). The percentage decrease ranged between a minimum of 4 percent to a maximum of 17 percent with an average of 12.7 percent considering simulations at all extrusion ratios (Fig. 9). The simulation results are highly encouraging to be subsequently applied to a real world extrusion process in the future after considering the product optimization.

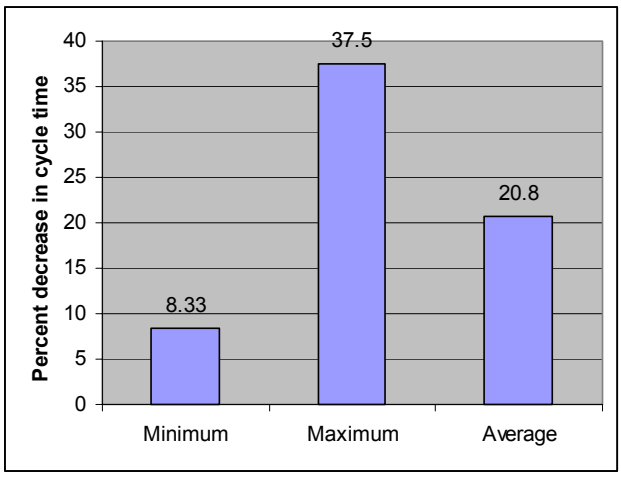


Figure 8: Graph showing percentage decrease in cycle time

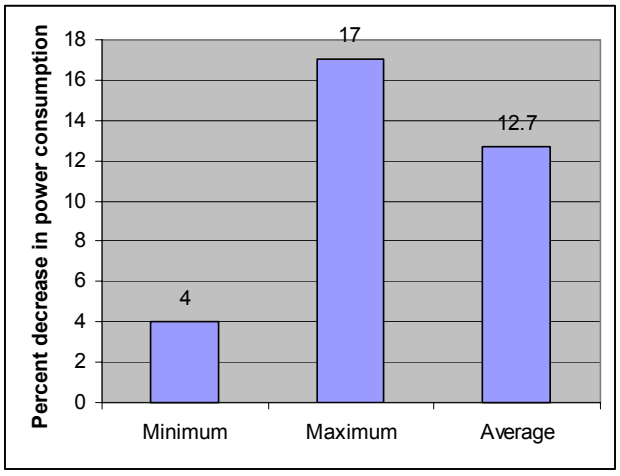


Figure 9: Graph showing percentage decrease in power consumption

DISCUSSIONS AND CONCLUSION

In this paper, the use of simulation for the optimization of principal extrusion variables was discussed. IGRIP and the dynamics module in IGRIP were used to model an actual extrusion machine. The model was then simulated using Graphical Simulation Language. The optimization obtained is positive and better optimization may be attained by considering wider factors that affect the dynamics of the extrusion process. Additionally, corresponding product optimization study will only better the actual optimization process and near the use of simulated data for real extrusion processes.

ACKNOWLEDGMENT

The authors would like to gratefully thank the KRF (Korea Research Foundation), BK21 (Brain Korea 21), and Digital Manufacturing Simulator and OLP System for Automobile Industry, South Korea for the fund that has been provided for this research.

REFERENCES

Banks, J., Handbook of Simulation, John Wiley and Sons Inc, New York, 1998.

Banks, J., Carson J. S., Nelson B.L., and Nicol D.M., Discrete event System Simulation, Prentice Hall, New Jersey, 2005.

Carson, J.S., "Introduction to Modeling and Simulation", Proceedings of the 2005 Winter Simulation Conference, ed. M.E. Kuhl, N.M. Steiger, F.B. Armstrong, and J.A. Joines, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Davis, J.R., Aluminum and Aluminum Alloys, ASM Specialty Handbook, ASM International, 1996

Delmia IGRIP Tutorial, Dassault Systems.

Hansson, S., "Simulation of Stainless Steel Tube Extrusion", Ph.D. Thesis, Department of Applied Physics and Mechanical Engineering , Division of Material Mechanics, Luleå University of Technology, Sweden, 2006

Lertsiriyothin, W. and Kumtib, M., "Simulation of flour flow in extrusion process by using computational fluid dynamics commercial software", ANSCSE, July 2004

Musselman, K.J., "Guidelines for Simulation Project Success", Proceedings of the 1994 Winter Simulation Conference, ed. J.D. Tew, S. Manivannam, D.A. Sadowski, and A.F. Seila, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Rao, P.N., Manufacturing Technology – Foundry, Forming and Welding, Tata McGraw Hill, New Delhi, 1995

Salazar, C.A.G., "Process simulation and training: the case of plastics extrusion", Journal of Modeling and Simulation in Materials science and Engineering, 1994

Sivaprasad, P.V., Venugopal, S, Davies, C.H.J., and Prasad, Y.V.R.K., "Identification of optimum process parameters for hot extrusion using finite element simulation and processing maps.", Journal of Modeling and Simulation in Materials Science and Engineering, Institute of Physics Publishing, Feb. 2004

Tibbetts, B.R., and Ting-Yung, J., "Extrusion Process Control: Modeling, Identification, and Optimization", IEEE transactions on control systems technology, vol. 6, no. 2, March 1998

AUTHOR BIOGRAPHIES

SACHIN M. BAJIMAYA was born in Nepal and went to South Korea to Ajou University in Suwon, where he is studying Industrial Engineering. Currently, he is a Graduate Research Assistant at the Automatic Monitoring and Control Laboratory. His e-mail address is: sachin@ajou.ac.kr.

CHANG M. PARK has a PhD in Industrial Engineering. Currently he is doing his post-doctoral research in the Automatic Monitoring and Control Laboratory in Ajou University, Suwon, South Korea.

GI-NAM WANG is a professor and Head of Department at the Department of Industrial and Information Systems Engineering, Ajou University, Suwon, South Korea. He holds a Ph.D. from Texas A&M University, USA. He has been a visiting professor at the University of Texas at Austin and is a member of many professional engineering societies. Web-page: http://madang.ajou.ac.kr/~gnwang/

A Comparison of Scheduling Algorithms for Multiprocessortasks with Precedence Constraints

Jörg Dümmler, Raphael Kunis and Gudula Rünger
Chemnitz University of Technology
Department of Computer Science
09107 Chemnitz, Germany
E-mail: {djo,krap,ruenger}@cs.tu-chemnitz.de

Keywords— **Multiprocessortask Programming, Scheduling, Distributed Memory, Scalable Computing**

Abstract— **Many parallel applications from scientic computing show a modular structure and are therefore suitable for the multiprocessortask programming model with precedence constraints. This programming model has been shown to yield better results than a pure data-parallel or a pure taskparallel execution on distributed memory platforms in many cases. The efficient execution of multiprocessortask programs requires an appropriate schedule, which takes the structure of the application and the performance characteristics of the target platform into account. Many heuristics and approximation algorithms have been proposed to fulfil this scheduling task. In this paper we consider popular scheduling algorithms that have been implemented in a scheduling toolkit. Specifically, we introduce Allocation-and-Scheduling-based algorithms and compare their runtime for large task graphs consisting of up to 1000 nodes and target systems with up to 256 processors. Furthermore we consider the quality of the produced schedules and derive a guideline describing which scheduling algorithm is most suitable in which situation.**

I. Introduction

A current challenge in the development of parallel applications for distributed memory platforms is the achievement of a good scalability even for a high number of processors. Often the scalability is impacted by the use of collective communication operations like broadcast operations, whose runtime exhibits a logarithmic or even linear dependence on the number of participating processors. Especially the advent of large homogeneous cluster systems and hierarchical cluster-of-clusters, which consist of multiple homogeneous clusters, necessitates a programming model that can help to reduce the communication overhead.

A possible approach is the model of multiprocessortask (short M-Task) programming with dependencies [7]. In this programming model a parallel application consists of a set of M-Tasks, which can be executed on an arbitrary subset of the available processors. Additionally there may be dependencies between M-Tasks meaning that these M-Tasks have to be executed one after another. These dependencies usually arise from communication phases that are necessary between the execution of M-Tasks to exchange data. For independent M-Tasks a consecutive or a concurrent execution on disjoint subsets of the available processors is possible. A schedule assigns each M-Task at least one processor and fixes the execution order of the M-Tasks. For a given M-Task program many different schedules may be possible. Which schedule achieves the best results, i.e. leads to a minimum parallel runtime of the application, depends on the application itself and on the target platform. Therefore for target platforms with different computation and communication behavior different schedules may lead to a minimum runtime. Determining the optimal schedule is an NP-hard problem, but many scheduling heuristics and approximation algorithms have been proposed to get a near optimal solution to this problem.

Developing an M-Task application is more complex and error-prone compared to the development of pure SPMD applications. This mainly results from two different types of communication (between M-Tasks vs. within an M-Task) and from the additional code required to manage the partitioning of the set of processors into subsets, on which the M-Tasks are executed. Furthermore, changes in the schedule of an M-Task application usually require a complex restructuring of the whole program resulting from a different processor group layout and a different communication pattern. A variety of languages, tools, libraries and frameworks to assist in the development of M-Task applications has been proposed. An overview is given in [1]. Most of these approaches still require the developer to manually specify the schedule for an application. As different target platforms may require different schedules this leads to a poor portability.

Many of the proposed scheduling algorithms for M-Task applications with precedence constraints have similarities on how to approach the scheduling problem. We distinguish three main categories, which we call *Allocation-and-Scheduling-based* algorithms, *Layer-based* algorithms, and *Configuration-based* algorithms. *Allocation-and-Scheduling-based* algorithms consist of an allocation step, which fixes the number of executing processors for each M-Task, and a scheduling step, which determines the execution order of the tasks and the exact processor groups. *Layer-based* algorithms shrink and decompose the directed acyclic graph representing an M-Task application into a set of layers of independent M-Tasks. The scheduling is performed for each layer in isolation and the resulting layer schedules are joined into a global schedule for the whole application. *Configuration-based* algorithms are single step methods that construct schedules based on a predefi-

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

ned set of possible configurations for each M-Task.

In this paper we examine *Allocation-and-Scheduling-based* algorithms. We present a detailed comparison of the runtime and the quality of the produced schedules for scheduling algorithms from this class. Finally, we derive a guideline, which M-Task scheduling algorithm is most suitable in which situation. The performance of M-Task scheduling algorithms was compared in [4], [5], however only small applications were considered and a different set of algorithms was used. To enable an automatic scheduling of M-Task applications based on cost expressions for the M-Tasks and for the communication between the tasks we develop a scheduling toolkit[2]. This toolkit includes scheduling algorithms from different categories including the presented *Allocation-and-Scheduling-based* algorithms. The use of a toolkit permits the examination of different scheduling algorithms using an identical environment by utilising similar data structures for representing M-Task applications and schedules.

This paper is structured as follows. Section II explains the multiprocessortask programming model with dependencies. Section III gives an overview of the considered scheduling algorithms. The obtained benchmark results are discussed in Section IV and Section V concludes the paper.

II. Programming Model

Many programming models are based on M-Tasks. In the M-Task programming model with precedence constraints a parallel application can be represented by an annotated direct acyclic graph (M-Task dag) $G = (V, E)$. Figure 1 shows an example of an M-Task dag.

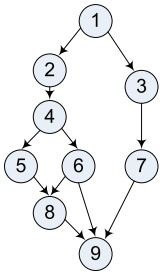


Fig. 1. Example of a small M-Task dag.

A node $v \in V$ corresponds to the execution of an M-Task, which is a parallel program part that can be executed on any nonempty subset $g_v \subseteq \{1, \ldots, P\}$ of the available processors of a P processor target platform. The size of a processor group $|g_v|$ is also denoted as the allocation a_v of a task v. The allocation $A : V \rightarrow [1, \ldots, P]^{|V|}$ of the M-Task dag unites the single allocations $a_v \; \forall v \in V$.

A directed edge $e = (v_1, v_2) \in E$ represents precedence constraints between the M-Tasks v_1 and v_2, i.e. v_1 produces output data required for v_2 and therefore v_1 and v_2 have to be executed one after another. Edges may lead to a data redistribution if the processor group changes, i.e. $g_{v_1} \neq g_{v_2}$ or if v_1 and v_2 require different data distributions. M-Tasks that are not connected by a path in the M-Task dag can be executed concurrently on disjoint subsets of the available processors. Each node $v \in V$ is assigned a computation cost $T_v : [1, \ldots, P] \rightarrow \mathbb{R}^+$ and each edge $e = (v_1, v_2) \in E$ is assigned a communication cost $T_{comm}(v_1, v_2)$.

The costs for a path in the M-Task dag under a given allocation are defined as the sum of the computing costs of all nodes and the data redistribution costs of all edges belonging to the path. The longest path in the M-Task dag is called the *critical path* and has a length of $T_{CP}(A)$. The set $CP(A)$ contains all nodes on the critical path. The *top level* $TL_v(A)$ of an M-Task $v \in V$ is the length of the longest path from any task without predecessors to task v excluding v. The length of the longest path from a node $v \in V$ to any node without successors including v is called the bottom level $BL_v(A)$. The work W_v of an M-Task $v \in V$ is the product of the computing time and the allocation, i.e. $W_v(a_v) = T_v(a_v) * a_v$. The average computing area T_A is the arithmetical mean of the works of all M-Tasks, i.e. $T_A(A) = \frac{1}{P} \sum_{v \in V} W_v(a_v)$.

An M-Task may either be a basic M-Task that is implemented using an SPMD programming style or a complex M-Task that is built up from other M-Tasks and can be represented by an M-Task dag. Hence a hierarchical structure as it is described in the TwoL(Two Level)-approach[8] arises.

The execution of an M-Task application is based on a schedule S, which assigns each M-Task $v \in V$ a processor group g_v and a starting time T_{S_v}, i.e. $S(v) = (g_v, T_{S_v})$. A feasible schedule has to assure that all required input data are available before starting an M-Task, meaning that all predecessor tasks have finished their execution and all necessary data redistributions have been carried out, i.e.

$$T_{S_n} + T_n(|g_n|) + T_{comm}(n, m) \leq T_{S_m}$$
$$\forall n, m \in V \text{ and } (n, m) \in E.$$

Furthermore M-Tasks whose execution time interval overlaps have to run on disjoint processor groups, i.e. if

$$[T_{S_n}, T_{S_n} + T_n(|g_n|)] \; \cap \; [T_{S_m}, T_{S_m} + T_m(|g_m|)] \neq \emptyset$$
$$\text{then } \; g_n \cap g_m = \emptyset \; \forall \; n, m \in V.$$

The makespan $C_{max}(S)$ of a schedule S is defined as the point in time at which all M-Tasks of the application have finished their execution, i.e.

$$C_{max}(S) = \max_{v \in V}(T_{S_v} + T_v(|g_v|).$$

In this paper we only consider non-hierarchical M-Task applications, i.e. only basic M-Tasks are available, and we do not take data redistribution costs into account. This is feasible as these costs are usually a magnitude lower compared to the computational costs of the M-Tasks and it is often possible to hide at least parts of these costs by overlapping of computation and communication. We only consider M-Tasks graphs that belong to the class of series parallel graphs (sp-graphs), because these graphs reflect the regular structure of most scientific applications. Series-parallel-graphs are a subset of directed acyclic graphs that are built by the following recursive definition [9]. A single node is an sp-graph. Two sp-graphs $SP_1 = (V_1, E_1), SP_2 = (V_2, E_2)$, can be combined to a new sp-graph by a series composition or a parallel composition. A sink node of a

sp-graph is a node without successors and a source node is a node without predecessors. A series composition connects every node from the set of sinks $T_1 \subseteq V_1$ of SP_1 with nodes from the set of sources $S_2 \subseteq V_2$ of SP_2 by a new edge: $SP_{new} = (V_1 \cup V_2, E_1 \cup E_2 \cup T_1 \times S_1)$. A parallel composition merges the set of nodes and the set of edges of SP_1 and SP_2 to a new series-parallel graph: $SP_{new} = (V_1 \cup V_2, E_1 \cup E_2)$.

III. Scheduling Algorithms

A popular approach to the M-Task scheduling problem with precedence constraints is a two-step approach introduced in [6] consisting of an allocation step and a scheduling step, which we therefore call *Allocation-and-Scheduling-based* algorithms. The allocation step determines the allocation a_v for each node $v \in V$ of the M-Task dag. The exact layout of the processor group g_v and the starting time index T_{S_v} is determined in the scheduling step. Most of these algorithms only differ in the allocation step and use a *modified List-Scheduling* algorithm in the scheduling step.

The modified List-Scheduling algorithm is based on a priority queue with different priority functions (earliest start time, bottom level, top level, smallest task or largest task). The List-Scheduling algorithm works as follows. Initialize the priority queue q by adding the source nodes of the M-Task dag. While q is not empty remove and schedule the head node v of the queue. The scheduling of v with an actual start time of T_{S_v} is done by finding a suitable set g_v of processors for a_v with the earliest processor ready time T_{g_v}. Afterwards the schedule time index T_{S_v} is re-evaluated by the following equation:

$$T_{S_v} = max(T_{g_v}, T_{S_v})$$

The finish time $T_{F_v} = T_{S_v} + T_v(a_v)$ of v is computed and the earliest start time of the successors of v and the processor ready times of the processors in g_v are set to this time. If the scheduling of v leads to the fulfilment of all precedence constraints of a successors of v this successor is added to the priority queue q. The worst case complexity of the modified List-Scheduling algorithm is $\mathcal{O}(E + Vlog(V) + VP)$ resulting from $\mathcal{O}(V + E)$ for the computation of the task priorities, $\mathcal{O}(VlogV)$ for removing V tasks from the queue and maintaining the queue ordering and $\mathcal{O}(VP)$ to schedule V tasks on P processors.

In the following we present the main ideas of scheduling algorithms belonging to the class of *Allocation-and-Scheduling-based* algorithms.

a) **Dataparallel:** The *Dataparallel* scheduler allocates all available processors to each M-Task in the allocation phase resulting in an SPMD processing style. A topological sort of the task graph can be used to obtain an order of execution of the tasks in the scheduling step. As the nodes of the input task graphs in our scheduling toolkit are already stored in topological order, the *Dataparallel* scheduler runs in $\mathcal{O}(V)$.

b) **Taskparallel:** The *Taskparallel* scheduler produces a schedule as it is known from uniprocessortask scheduling. The allocation phase assigns each M-Task a single processor and a List-Scheduling algorithm with the bottom levels as a priority function is used for the scheduling phase. The worst case time complexity of $\mathcal{O}(E + VlogV + VP)$ for the *Taskparallel* scheduler is equal to the List-Scheduling algorithm.

c) **TSAS:** The *Two Step Allocation and Scheduling(TSAS)*[6] scheduler transfers the problem to find a discrete allocation for each M-Task to an optimization problem in the continuous space. The objective of the optimization is to find an allocation $A_c : V \rightarrow \mathbb{R}^{|V|}$ that minimizes $\max\{T_{CP}(A_c), T_A(A_c)\}$. The intention is to find an allocation that is a good trade-off between critical path length and average area, which are both lower bounds on the makespan of any feasible schedule. If the cost functions are posynomials a convex optimization problem results, which has a unique global minimum that can be determined by an iterative algorithm in polynomial time. A posynomial function f of a positive vector variable $x \in \mathbb{R}^m$ has the form

$$f(x) = \sum_{i=1}^{N} c_i \prod_{j=1}^{m} x_j^{a_{ij}}$$

with positive coefficients $c_i \in \mathbb{R}^+$ and exponents $a_{ij} \in \mathbb{R}$. For our tests we use cost expressions based on Amdahl's law, which are posynomial functions. The result of the allocation phase is obtained by mapping the continuous solution of the optimization problem to discrete space. The scheduling phase uses a List-Scheduling algorithm with the earliest possible start time of a task as a priority function.

d) **CPA:** The *Critical Path and Area-based scheduling (CPA)*[5] scheduler was designed as a low-cost scheduler and a computationally cheap heuristic is employed for the allocation phase. The idea of the allocation phase is to find an allocation A that minimizes $\max\{T_{CP}(A), T_A(A)\}$ and is therefore similar to *TSAS*. The starting point is an allocation of one processor per task, i.e. $a_v = 1 \; \forall v \in V$. Each iteration of the main iteration loop chooses a critical path task $v \in CP(A)$ and increases its allocation, i.e. $a_v = a_v + 1$. As a consequence, the critical path may change and other nodes are considered in subsequent iterations. The main loop terminates when the average area exceeds the length of the critical path for the current allocation, i.e. $T_A(A) \geq T_{CP}(A)$. The scheduling phase uses a List-Scheduling heuristic with the bottom levels of the tasks as a priority function. The worst case complexity of *CPA* is $\mathcal{O}(V(V + E)P)$, which arises from $\mathcal{O}(VP)$ iterations of the main iteration loop each requiring $\mathcal{O}(V + E)$ time to compute the critical path and a single execution of the List-Scheduling algorithm.

e) **CPR:** The *Critical Path Reduction (CPR)*[4] scheduler follows a similar approach as the *CPA*-scheduler but uses a more complex heuristic in the allocation phase to reduce the length of the critical path in the M-Task dag. *CPR* starts with an allocation of one processor per task, i.e. $a_v = 1 \forall v \in V$. The main iteration loop first computes a priority *prio* for

all tasks based on the sum of the top and bottom level, i.e. $prio_v = TL_v + BL_v \forall v \in V$ and inserts all tasks in a priority queue. Afterwards the head task v of the priority queue is removed from the queue, the allocation is increased by 1, i.e. $a_v = a_v + 1$, and a List-Scheduler with a bottom level priority function is run with the current allocation. If the constructed schedule has a lower makespan than any previously obtained schedule, the current allocation is commited and the main loop is started over. Otherwise the changes are rejected, i.e. $a_v = a_v - 1$, and the next task in the priority queue is considered. The main loop terminates, when the priority queue runs empty, i.e. there is no task for which increasing its allocation results in a better schedule. The time complexity of *CPR* is $\mathcal{O}(EV^2P + V^3Plog(V) + V^3P^2)$ and results from $\mathcal{O}(VP)$ executions of the main iteration loop in the worst case, which occurs when P processors are allocated to each of the V tasks. Each loop iteration may have to call the List-Scheduling algorithm for all V tasks in the worst case.

f) **MSAA:** The *Modified sp-graph approximation algorithm (MSAA)*[3] scheduler uses an approximation algorithm based on integer values for the execution time of the tasks in the allocation phase and an earliest start time List-Scheduler in the scheduling phase. The approximation algorithm tries to decide within pseudo-polynomial time whether an allocation with costs $c(A) \leq X$ exists for a given positive integer bound X. X represents the critical path length in the M-Task dag for a pure taskparallel allocation

$$c(A) = max\{T_{CP}(A), T_A(A)\} \leq C_{max}(S).$$

This algorithm operates not at the sp-graph itself but on the decomposition tree of the sp-graph. The sp-graph decomposition tree $G_D = (V_D, E_D)$ corresponds to a rooted, ordered, binary tree [9]. The internal nodes correspond to the composition of the sp-graph and are labeled s (series composition) or p (parallel composition). The leafs are the nodes in the sp-graph. After the decomposition of the sp-graph a matrix F of dimension $|V_D| \times X$ is built that contains the values $F[v_D, l], 1 \leq v_D \leq |V_D|, 0 \leq l \leq X$. Each $F[v_D, l]$ represents the smallest possible value for the work $W_{v_D}(A_{v_D})$ of a node v_d in the decomposition tree that holds the following property: An allocation A for the tasks in the sub decomposition tree under v_d exists with

$$T_{CP}(A) \leq l, T_A(A) \leq W_{v_D}(A_{v_D}).$$

A dynamic programming approach is used to compute all $F[v_D, l]$ values starting in the leafs of the decomposition tree and moving upwards to the root. The last step of the algorithm is to find a value for l with $F[root_D, l]/P \leq X$. Because more than a single l can be found, we use all possible candidates of l in the list scheduling step to determine the best solution. The allocations for the l-values can be found by storing additional information in the computing step of all values in F. The complexity of the allocation algorithm is $\mathcal{O}(|V_D| * P * X^2)$. X^2 is the main factor in the complexity. We try to decrease the runtime of the algorithm by mapping the runtimes of the nodes to integers getting an X that is small but produces relative good schedules. The mapping has to be performed, because the algorithm needs integers to process the dynamic programming approach. We use the following formula to map the runtimes: $X = (1+|V|/adapt)*4, adapt = 250$. This mapping depends on the number of nodes $|V|$ in the sp-graph. It tries to find a good solution for getting different integer values for different original execution times of the tasks by keeping X small. The value for *adapt* is based on runtime tests on our target machine and it is possible to find a good solution for *adapt* by running some tests on other target machines.

IV. Results

A. Testing Environment

The benchmark tests presented in this Section are obtained by running the scheduling toolkit on an Intel Xeon 5140 ("Woodcrest") system clocked at 2.33 GHz. The available main memory was 8 GB cached by an L2 cache with a size of 4 MB. To run the scheduling toolkit the 64 bit version of the *Java 2 SE Runtime Environment (JRE)* Version 5.0 Update 9 was used.

For the benchmarks, we use *test sets* consisting of 100 different M-Task dags, which belong to the class of series-parallel-graphs (sp-graphs). The generation algorithm used to construct these graphs starts with a number of sp-graphs consisting of a single node and randomly combines these graphs by a serial or a parallel composition. Afterwards all nodes of the graph are annotated by a runtime estimation formula according to Amdahl's law ($T_{par} = (f + (1-f)/p) * T_{seq}$), which describes the parallel runtime T_{par} on p processors for a problem with an inherent sequential fraction of computation $f (0 \leq f \leq 1)$ and a runtime on a single processor $T_{seq} (T_{seq} > 0)$.

B. Runtime Results

In this Subsection we consider the runtime of the implemented scheduling algorithms. All presented measurements are the arithmetical mean of the runtimes for each M-Task dag within a test set. The same test set was used when varying the number of processors, but changing the number of nodes requires a different set.

The *Dataparallel* and *Taskparallel* schedulers achieve the lowest runtimes of all scheduling algorithms as the execution does not involve a sophisticated scheduling process. The runtimes of the *Dataparallel* scheduler range from 0.5 ms for 50 nodes to 10.4 ms for 1000 nodes and are independent from the number of processors. These execution times can be considered as a general overhead factor for the management of the internal structures of the scheduling toolkit. Figure 2 shows the average runtimes of the *Taskparallel* scheduler. The measurements for the *Taskparallel* scheduler show an almost linear dependence on the number of nodes and a slow increase of the runtime with the number of processors. The runtime for 256 processors is about 20%

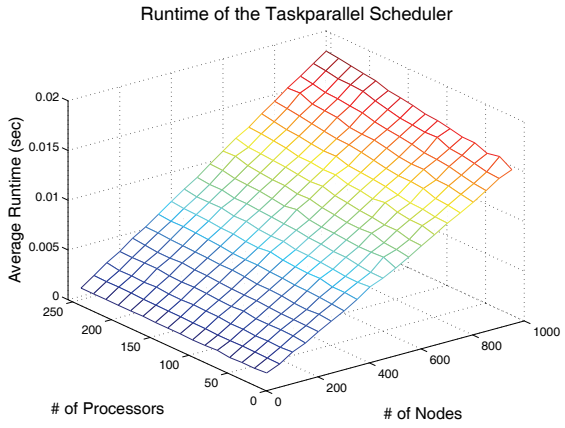


Fig. 2. Average runtime of the *Taskparallel* scheduler for varying number of nodes and processors.

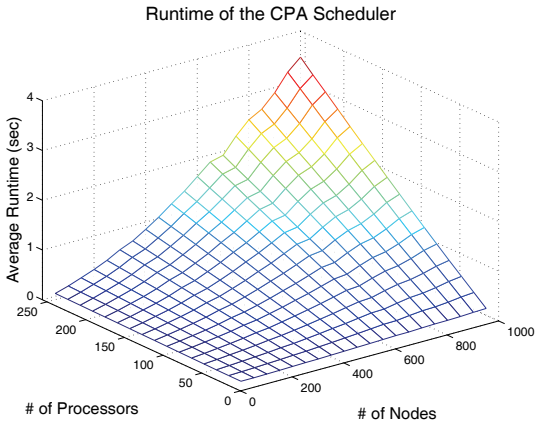


Fig. 4. Average runtime of *CPA* dependent on the number of nodes and processors.

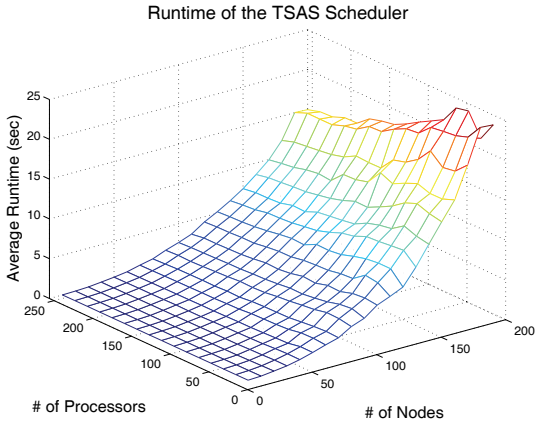


Fig. 3. Average runtime of *TSAS* for different number of nodes and processors.

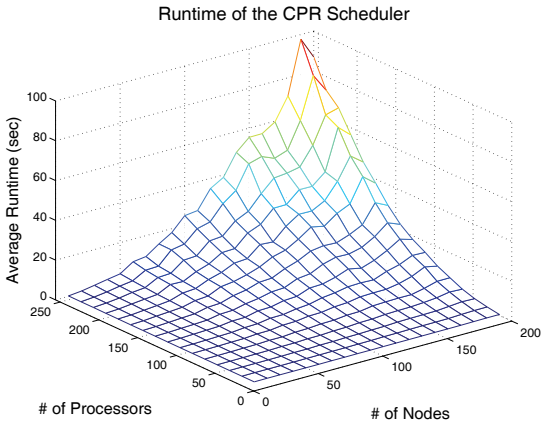


Fig. 5. Average runtime of *CPR* for varying number of nodes and processors

higher compared to the execution time for 16 processors. These runtimes are also a measure for the List-Scheduling part in *CPA*, *CPR*, *TSAS* and *MSAA*, as the same heuristic is employed in these algorithms.

The average runtimes of *TSAS* are shown in Figure 3. Because of the high runtime requirements for 100 test runs we only show results for up to 200 nodes for this scheduling algorithm. The main part of the work of *TSAS* is done in the allocation step, when solving the convex optimization problem. The runtime of this step is directly influenced by the number of required iterations. For all tested numbers of nodes the average runtime of *TSAS* decreased when increasing the number of processors. This is an interesting behavior that is unique within the tested scheduling algorithms and can be explained by a faster convergence of the convex programming approach. In our tests we experienced a medium deviation between the minimum and the maximum runtimes of *TSAS* for a given test set.

As the average execution times of *CPA* that are presented in Figure 4 show, *CPA* achieves a good performance even for a high number of nodes for target platforms with a low number of processors. The runtime increases linearly with the number of processors agreeing with the worst case complexity of *CPA*. Although the results show a constant increase of the runtime with the number of nodes and the number of processors, the runtimes within a given test set exhibit a large deviation. This mainly results from a different number of iterations in the main allocation loop, which is responsible for the biggest part of the runtime. The number of iterations can be between 1 (if the termination criterion is met with the initial allocation) and $\mathcal{O}(VP)$ in the worst case (when the resulting allocation assigns all processors to each node).

The average runtime results for *CPR* are shown in Figure 5. *CPR* exhibits the slowest average runtimes of all implemented scheduling algorithms and was therefore only tested for M-Task dags with up to 200 nodes. Compared to *CPA* the much higher runtime results from the execution of the List-Scheduling heuristic in each iteration, whereas *CPA* only requires a single List-Scheduling step. For target platforms with a low number of nodes *CPR* still achieves reasonably low runtimes. The runtime of *CPR* strongly depends on the input scheduling problem. The deviation between the minimum and maximum runtimes for a given test set is the highest of all scheduling algorithms. The reason for this behavior is similar to *CPA* a varying number of iterations in the main allocation loop, which can be between 1 and $\mathcal{O}(VP)$. This can also explain, why the test set with 190 nodes requires a higher average run-

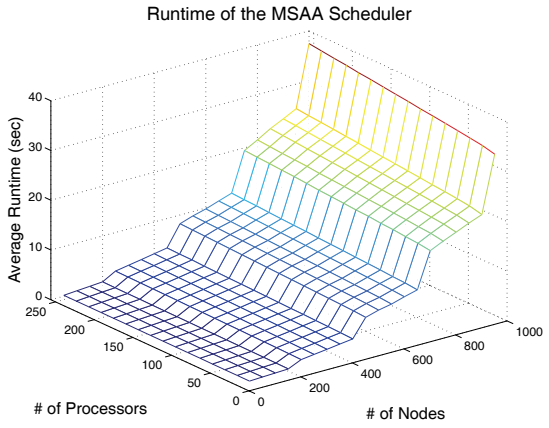


Fig. 6. Average runtime of *MSAA* for varying number of nodes and processors

TABLE I: Average Runtimes relative to the *Dataparallel* Scheduler.

Scheduler	200 nodes 16 proc.	200 nodes 256 procs.	1000 nodes 16 procs.	1000 nodes 256 procs.
Data	1	1	1	1
Task	1.36	1.67	1.41	1.71
TSAS	11484	7254	n/a	n/a
CPA	9.00	158	13.1	331
CPR	214	42215	n/a	n/a
MSAA	221	376	3126	3573

time compared to the test set with 200 nodes.

Figure 6 shows the runtime results for *MSAA* with critical path adaption. The results show that this algorithm achieves a good performance for small and high numbers of processors and nodes in comparison to *CPR* and *TSAS*. The dependency on the X-value can be seen in the jumps at 250, 500, 750 and 1000 nodes. The runtime is linear in the number of nodes and linear in the number of processors if X is fixed. This linear runtime results from the structure of the $|V_D| \times X$ matrix used for computing the F-values that is linear in $|V_D|$ and P and the main factor is X^2 in the allocation step. The runtimes within a given test set exhibit only small deviations. This results from the dependency of the runtime on X^2 which is much larger than $|V|$ and P.

Table I gives an overview of the relative runtimes of all tested scheduling algorithms compared to the *Dataparallel* scheduler averaged over 100 test runs. It comes to no surprise that the *Taskparallel* and *Dataparallel* Schedulers have a much lower runtime compared to the other algorithms as no sophisticated scheduling logic is involved. From the specialized scheduling algorithms *CPA* achieves the highest performance and especially for a low number of processors clearly outperforms all other algorithms. For a high number of processors the gap between *CPA* and *MSAA* becomes smaller and it can be assumed that *MSAA* beats *CPA* for processor numbers somewhat higher than 256. *TSAS* and *CPR* (for a high number of nodes) exhibit a considerably higher runtime than all other algorithms. *CPR* is faster than *TSAS* for a low number of processors, whereas *TSAS* beats *CPR* for larger target platforms.

If the runtime of the scheduling algorithm is an issue,

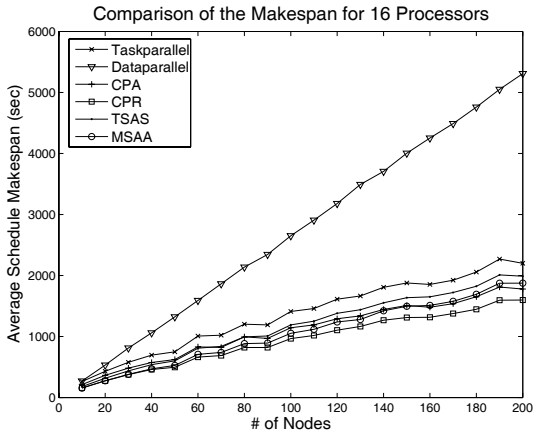


Fig. 7. Comparison of the average makespan of different scheduling algorithms for task graphs with 10 to 200 nodes and 16 available processors.

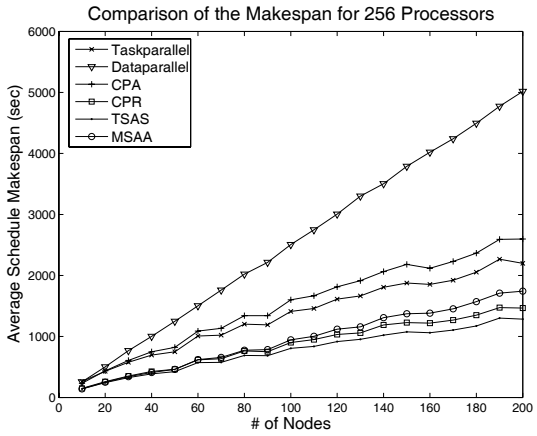


Fig. 8. Comparison of the average makespan of different scheduling algorithms for task graphs with 10 to 200 nodes and 256 available processors.

CPA is a good choice, because it achieves the lowest runtimes of all specialized scheduling algorithms. For target systems with many processors *MSAA* might be a good choice since its runtime is also almost independent from the structure of the input problem.

C. Quality of the Schedules

In this Section we consider the quality of the produced schedules, i.e. the makespan based on the runtime estimation formulas. First we consider the average makespan for the test sets with up to 200 nodes. The results for all tested scheduling algorithms are shown in Figure 7 for 16 available processors and in Figure 8 for 256 processors respectively. Especially Figure 8 shows no crossing point between the curves of the scheduling algorithms, i.e. a scheduling algorithm that achieves a

TABLE II: Speedups of the produced schedules relative to the *Dataparallel* Scheduler.

Scheduler	16 procs.	64 procs.	128 procs.	256 procs.
Data	1	1	1	1
Task	1.89	1.81	1.79	1.79
TSAS	2.25	2.73	2.98	3.14
CPA	2.33	1.91	1.72	1.60
CPR	2.75	2.78	2.79	2.80
MSAA	2.51	2.68	2.65	2.62

TABLE III: Number of constructed schedules with the lowest makespan for 16 (left value) and 256 processors (right value).

Scheduler	50 nodes	100 nodes	150 nodes	200 nodes
Data	0/0	0/0	0/0	0/0
Task	0/0	0/0	0/0	0/0
TSAS	1/68	0/87	0/93	0/92
CPA	9/0	11/0	12/0	18/0
CPR	67/18	77/8	80/7	80/8
MSAA	23/14	12/5	8/0	2/0

TABLE IV: Recommended scheduling algorithms for different situations.

	low number of processors	high number of processors
low number of nodes	*CPR** *CPA***	*TSAS** *MSAA***
high number of nodes	*CPR** *CPA***	*TSAS** *MSAA***

* best quality ** good quality, reasonable runtime

better quality for a low number of nodes is also better for a higher number of nodes. The *Dataparallel* scheduler delivers the schedules with the highest makespans for all tested problem instances. The gap to the other scheduling algorithms increases with the number of nodes as more options for a mixed task and data parallel execution are available. Table II lists the speedup of all scheduling algorithms averaged over all task graph sizes relative to the *Dataparallel* scheduler. The results of all other scheduling algorithms lie closer together for 16 available processors and range from a speedup of 1.89 (*Taskparallel*) to a speedup of 2.75 (*CPR*). As Figure 7 shows, *CPR* constantly achieves the best average quality for 16 available processors. On the other hand *CPR* gets outperformed by *MSAA* in 18%, by *CPA* in 12% and by *TSAS* in 4% of the test cases.

For 256 available processors the schedules with the lowest average makespan are delivered by *TSAS* with a speedup of 3.14 compared to a dataparallel execution, followed by *CPR* with a speedup of 2.8. The average results obtained by *MSAA* are located in the mid range and *CPA* exhibits an unusual behavior. The makespan of the schedules delivered by *CPA* increases if more processors are available for most scheduling problems. For 16 processors *CPA* could achieve competitive makespans but is worse than the *Taskparallel* scheduler for 256 processors.

Table III shows for each scheduling algorithm the number of times it could generate the schedule with the lowest makespan for different number of nodes. The left number corresponds to 16 available processors and the right number belongs to 256 available processors. The *Dataparallel* and *Taskparallel* schedulers never construct a minimal schedule. As the average results already showed, *CPR* obtains the best results for 16 processors and *TSAS* has the lead for 256 processors.

In summary the results state that there is no scheduling algorithm that clearly dominates the test field. Our results for low numbers of processors agree with the findings from [5], [4], where *CPR* achieves the best quality and *CPA* was shown to be competitive to other scheduling algorithms. We have shown that for a high number of processors the quality of *CPA* gets worse and *CPR* is mostly outperformed by *TSAS*. Especially the experiments in this paper show that the M-Task programming approach clearly outperforms a pure dataparallel execution and is therefore a suitable model for parallel computation. Table IV gives an overview of the suggested scheduling algorithm depending on the size of the input problem (number of nodes), the number of processors and the runtime of the scheduling algorithm.

V. Conclusion and Future Work

In this paper we have evaluated a variety of scheduling algorithms for M-Tasks with dependencies belonging to the class of *Allocation-and-Scheduling-based* algorithms. We compared the runtime of the scheduling algorithms and the makespans of the generated schedules. If the makespan of the resulting schedule should be minimized, *CPR* is a good choice for a low number of processors and the runtime is reasonably low in this case. For a high number of processors *TSAS* is faster and constructs better schedules compared to *CPR*. As a schedule for an application usually depends on the input data size and has therefore to be recomputed multiple times, the running time of the applied scheduling algorithm becomes an important issue. In this case *CPA* is a good choice for a low number of processors, but the resulting schedules are not competitive for a high number of processors. *MSAA* offers a good scalability for high numbers of nodes and processors and produces schedules with a middle-ranked makespan.

Future work includes the examination of additional categories of scheduling algorithms. It is planned to make the algorithms that are implemented in the toolkit available to other applications in form of a library.

References

[1] H. Bal and M. Haines, "Approaches for integrating task and data parallelism," *IEEE Concurrency*, vol. 6, no. 3, pp. 74–84, 1998.

[2] J. Dümmler, R. Kunis, and G. Rünger, "A Scheduling Toolkit for Multiprocessortask-programming with Dependencies," *submitted for publication.*

[3] R. Lepere, D. Trystram, and G. J. Woeginger, "Approximation algorithms for scheduling malleable tasks under precedence constraints," *Lecture Notes in Computer Science*, vol. 2161/2001, 2001.

[4] A. Radulescu, C. Nicolescu, A. van Gemund, and P. Jonker, "CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems," in *IPDPS '01: Proc. of the 15th Int. Par. & Dist. Proc. Symp.* IEEE Computer Society, 2001, p. 39.

[5] A. Radulescu and A. van Gemund, "A Low-Cost Approach towards Mixed Task and Data Parallel Scheduling," in *Proc. of the 2001 Int. Conf. on Parallel Processing.* IEEE Computer Society, 2001, pp. 69–76.

[6] S. Ramaswamy, S. Sapatnekar, and P. Banerjee, "A framework for exploiting task and data parallelism on distributed memory multicomputers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 11, pp. 1098–1116, 1997.

[7] T. Rauber and G. Rünger, "Compiler support for task scheduling in hierarchical execution models," *J. Syst. Archit.*, vol. 45, no. 6-7, pp. 483–503, 1998.

[8] ——, "A Transformation Approach to Derive Efficient Parallel Implementations," *IEEE Trans. on Software Engineering*, vol. 26, no. 4, pp. 315–339, 2000.

[9] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The recognition of series parallel digraphs," Tech. Rep., 1979.

OPTIMIZING CACHE EFFICIENCY BY SIMULATION DRIVEN AUTOMATIC PADDING

Marco Höbbel, Thomas Rauber, Carsten Scholtes
Fachgruppe Informatik
Universität Bayreuth
Universitätsstr. 30
95447 Bayreuth
{hoebbel,rauber,carsten.scholtes}@uni-bayreuth.de

ABSTRACT

We present a toolset to automatically optimize the cache efficiency of an arbitrary application by dynamically padding memory allocations. The toolset is also suitable to guide manual optimizations. Histograms are used to evaluate cache simulations of memory traces of the applications considered. A general algorithm is presented that calculates optimized pad sets based on the information contained in the histograms. These pad sets can then be used to optimize further runs of the applications examined. Experiments show that the cache hit rates of the modified applications are considerably increased.

Keywords: *cache optimization, dynamic padding, memory tra ces, performance visualization*

I. INTRODUCTION

For most programs, the execution time should be as short as possible. Especially for computation intensive applications a good runtime efficiency minimizes the throughput time, thereby preserving computation resources for other pending or scheduled applications. Scientific problem solvers, for example, are mostly time and data intensive in nature. Many mathematical computations like vector-matrix or matrix-matrix multiplications and additions in multiphase, iterative or cyclic algorithms refer to user data in problem specific access patterns. For regular applications, it can be expected that repeated program executions exhibit a similar memory access pattern. Many data intensive applications therefore benefit from a high memory bandwidth which in modern architectures is supported by supplying a multi-level cache hierarchy. The efficiency of caching depends strongly on temporal or spatial reuse, so potential conflicts should be avoided. Different techniques are applied in order to ensure that the fastest cache level in the memory hierarchy is addressed first. Padding is one such approach and has been successfully applied for many high performance applications. Unfortunately, most of these optimization techniques are problem specific and difficult to adapt for general use. Additionally to a high experience of the programmer, a deep analysis of the program and its access pattern is required to become aware of the relevant effects and to finally optimize cache usage. Due to the immense effort associated with this approach, often only post-programming optimizations by the compiler or the runtime system are employed for gaining high efficiency with minimal effort.

In order to support analyzing code sections optimized by hand and runtime optimizations of applications without having their source code, a trace based acquisition of data for analyzing the application's memory access pattern is suitable. We present such an approach in this paper. The advantage of the approach is (1) that it can be applied also for programs for which the source code is unknown and (2) that the programmer does not have to apply low-level code optimizations to obtain a good overall performance. The approach is based on histograms which are data structures having the potential to hold the necessary information like access patterns acquired during a first phase tracing program run. Based on the histograms of an application, a second phase computes a memory aligning pad set of all or at least the major cache impacting references which leads to a better overall performance. The generated optimal offsets can then be applied by a compiler to create an optimized binary code. In the case of only having a binary executable, an enhanced runtime system can intercept memory allocation operations and patch them to provide the optimized padding for the following program runs.

The paper's objective is the presentation of a post-compilation, performance optimizing and visualizing toolset, which is suitable for general use, even in the absence of the source code of the application to be optimized. In section II, we introduce the notation used in the rest of the paper. In section III we describe the optimization method. It is subdivided into three parts concerning the trace driven data acquisition (III-A), the histogram types describing conflicts (III-B) and the automated optimization algorithm (III-C). In section IV we present measured results for different example applications before we discuss related work and conclude our paper.

II. TERMINOLOGY

The memory interface of the execution platform is modeled as follows: The cache [Handy, 1998] is subdivided into *cache lines*, which represent the minimal amount of data transferable between cache and main memory. The main memory is subdivided into *memory lines* of the same size S_{CL}, which are mapped round-robin onto the cache lines. An access which can be satisfied directly from the cache is called a (cache) *hit* and takes a latency

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

of t_{L_H} CPU cycles. If the cache does not contain a copy of the memory line to be accessed, the access is called a (cache) *miss* and results in a maximal stalling time of t_{L_M} cycles for validating by accessing the main memory. An access to a memory line that has been accessed before is called a *reuse*. The last access to the reused memory line before the reuse is called the *source access* of the reuse. The time between source access and reuse is called the *reuse distance* of the reuse. The set of accesses occurring within the reuse distance of a reuse, thus, potentially replacing the memory line to be reused, is called *interference* of the reuse.
Many cache hardware platforms do not only validate the cache line just accessed, but also preload the next k ones in order to prevent latency cycles by adopting an anticipating early validation strategy [Oren, 2000]. A simple *k-line* prefetching strategy with $k = 1$ tries to also load the line in sequence of the last two accesses into the cache
A *reference* is a contiguous chunk of memory containing elements accessed by the application. From the address bus' point of view, each reference R is classified by its base address $base\_addr_R$ (the first address of the chunk) and the size of its elements $size_R$. For analyzing, we collect all addresses of the accesses of an array reference and store an extra attribute that counts the number $length_R$ of array elements contained. The program/data *trace* T is the set of all accessed data- and instruction-bus addresses. It contains all collected accesses in the form of tuples $ma = (time, addr, acctype) \in T$ specifying $time \in [0,$ runtime in cycles$]$ in absolute system cycles, the accessed address $addr \in [0,$ virtual memory size$)$ of the reference and the access type $acctype$, denoted by either *r* (data read), *w* (data write), *d* (data read or write) or *i* (instruction read).
Furthermore, a discriminator $\mathbf{Ref}^{acctype}_{refname}$ can be applied to T to extract the specific accesses of type $acctype$ of the reference $refname$:

$$\begin{aligned} Ref^t_R := \{ & ma = (time, adr, type) : \\ & adr \geq baseadr_R, ma \in T, type = t \\ & adr < baseadr_R + size_R \cdot length_R \} \subset T \end{aligned}$$

For simplicity, Ref^*_R includes all traced tuples of reference R. If, for example, only the data bus is used, it is $Ref^*_R = Ref^d_R$.

III. METHOD

A. Data acquisition

In a first phase, a trace of a fixed problem size is generated by capturing the addresses and properties of all memory accesses. Different tools can be used to generate such traces:

QPT: On SPARC [Mauro and McDougall, 2001], [SPARC, 2000] architectures, for example, there exists the QPT [Larus, 1993] toolset. It distinguishes between data read, data write and instruction read accesses. These traces are useful for binary level post-compilation optimizations. Such optimizations will rely on all the runtime information caught. They will be realized by manipulating the original binary to use dedicated dynamic memory allocations.

SPD: Given the source code of the application to be optimized, there is another method for acquiring trace data that supports a deeper analysis. It is based on **s**elf **p**rotocoled **d**atatypes (SPD) that we developed and that can be used instead of the original data types. Tracing with these is helpful for examining and refining parts of the whole program by focusing on only the basic conflicting scenarios within the source code. Additional access attributes like the name of the triggering reference (*refname*) and its data type (*field-type*) can be extracted without changing the binary used to produce the trace. These attributes may be taken into account by future optimization strategies like, for example, a symbolic reference analysis.

Other: Due to the availability of the GNU tool *gdb* on many architectures, it is interesting to use it for data acquisition by simulated program runs. Binaries (with symbol table) include valuable symbolic information helpful for generating more transparent analyses.
Another pure simulation tool *Simics* [Magnusson et al., 2002], [Magnusson et al., 1998] is useful for address trace generation with the ability to focus on special application bounded addresses only.

For these and other tools it is mandatory that the trace generation is not corrupting inter-reference correlations. Apart from analyzing given applications, the tracing tools are also useful to verify the benefits of optimizations.

B. Histograms

During the trace run of the program to be analyzed, the generated access pattern has to be stored in a way that conserves information on which the later optimization phase depends. Our simulation based histogram illustrations expose patterns in an intuitively understandable manner. A directed extraction of the data stored in the first phase supports the final automatic optimization step. Furthermore, the intuitive understandability of graphically visualized histograms inspires the invention of new types of histograms capturing additional information suitable to support a more target oriented or better performing analysis.
For simplicity, we consider in the following only data accesses T^d_* (both, reading and writing) and no instruction reading accesses. Our histograms count accesses to address offsets or address differences of accesses. They hold counts for up to H_{size} consecutive such address values. H_{size} will match the size S_C of the cache of the target platform. Different types of histograms are employed:

SRH – single reference histogram: The stored data of the first histogram type ${}_SH_R$ is concerned with a single reference R only. For an address $baseadr_R + i$ the value ${}_SH_R(i)$ counts the number of memory ac-

cesses to the address offset i of reference R, such that $\sum_{i=0}^{H_{size}-1} {}_SH_R(i) = |Ref_R^d|$. The latter represents the *significance* of the reference R in comparison to the other references in the program trace. The generated pattern stored in the histogram is evaluated by accumulating the offsets from the base address $baseadr_R$. The histogram ${}_SH_R$ is then defined as the set of all pairs of offsets i and their corresponding access count ${}_SH_R(i)$:

$$
\begin{aligned}
{}_SH_R(i) \quad &:= \quad |\{ma = (time, adr, type) : \\
&\qquad ma \in Ref_R^d, \\
&\qquad i = (adr - baseadr_R)\%H_{size}\}| \\
\\
{}_SH_R \quad &:= \quad \{(i, {}_S H_R(i)) : \\
&\qquad i \in [0, H_{size})\}
\end{aligned}
$$

For scalar 0-dimensional types of references, the histograms degenerate to simple counters.

SDH – self distance histogram: A self distance histogram ${}_DH_R$ refers to a single reference R, too, but accumulates the distances of consecutive accesses to R. We first introduce the predecessor $pred_{R_2}(ma) \in Ref_{R_2}^d$ of an access $ma = (t_a, a_a, t) \in Ref_{R_1}^t$ in order to then define ${}_DH_R(i)$ for an address distance i:

$$
\begin{aligned}
pred_{R_2}(ma) \quad &:= \quad (t_b, a_b, t) \in Ref_{R_2}^t \text{ with} \\
t_b \quad &= \max\{time : \\
&\qquad \exists\, (time, a, t) \in Ref_{R_2}^t, \\
&\qquad time < t_a\} \\
\\
{}_DH_R(i) \quad &:= \quad |\{ma = (t_a, a_a, d) \in Ref_R^d : \\
&\qquad \exists\, (t_b, a_b, d) = pred_R(ma), \\
&\qquad (a_a - a_b)\%H_{size} = i\}|
\end{aligned}
$$

The information extracted by this kind of histogram describes the step increment of sequential memory accesses. Other references with similar step increment patterns can be padded in order to avoid thrashing constellations. Furthermore, given a sequential access pattern, which can be observed for many references, the histogram allows us to detect the prefetch distance and to estimate the resulting usage efficiency for each cache line fetched. Larger distances imply a sparser usage. This information can be used to resolve padding conflicts with other references by favoring references with more efficient cache line usage.

PDH – pairwise distance histogram: This type of histogram calculates distances like SDH, but this time, the accesses refer to the consecutive accesses between two distinguished references R_1 and R_2:

$$
\begin{aligned}
{}_PH_{R_1,R_2}(i) := |\{&ma = (t_a, a_a, d) \in Ref_{R_1}^d : \\
&\exists\, (t_b, a_b, d) = pred_{R_2}(ma), \\
&((a_a - baseadr_{R_1}) - \\
&(a_b - baseadr_{R_2}))\%H_{size} = i\}|
\end{aligned}
$$

The histogram ${}_PH_{R_1,R_2}$ highlights distances conflicting spatially. To avoid the problem of mutual thrashing, all significant distances in ${}_PH_{R_1,R_2}$ should be excluded in the later pad set.

$sList$ =sort(${}_SH_*^d$ by $|\,{}_SH^d|$)
${}_gH = H_{stack}^d$
FOR $H \in sList$ DO
    offsets$[H] = f_{min}({}_gH, H, \text{offsets})$
    ${}_gH = {}_gH \oplus (H \gg \text{offsets}[H])$
DONE

Figure 1: Optimization Algorithm

Histograms support a number of basic operations useful during the optimization. Two histograms H_1 and H_2 can be *merged* to a new histogram $H = H_1 \oplus H_2$ which is defined a follows:

$$\forall_{i=0}^{H_{size}-1} H(i) = H_1(i) + H_2(i)$$

A histogram H can be *rolled* by an offset o to yield a new histogram $H' = H \gg o$ which is defined as follows:

$$\forall_{i=0}^{H_{size}-1} H'((i+o)\%H_{size}) = H(i)$$

The *significance* $|H|$ of a histogram H is defined as the number of the accesses contained:

$$|H| := \sum_{i=0}^{H_{size}-1} H(i)$$

In the following, these operations and constructs are used to formulate the optimization algorithm.

C. Optimization

The third and final phase is the optimization. It uses the different types of histogram data generated to determine a pad set for a later cache optimized program execution. The algorithm is based on a greedy strategy recognizing the patterns the histograms describe. Appropriate heuristics consider the conflict potential and find a padding with minimal miss potential.

The algorithm we propose, as outlined in Figure 1, is straightforward and tries to simplify the complex problem of finding the best fitting overall solution by using reference and inter-reference specific histogram overlaying. To do so, (1) it applies the greedy paradigm to a sorted list of histograms which are padded with a "highest significance first" strategy. Then, the reference chosen to be padded is shifted by an offset that offers the best overlay with a global histogram ${}_gH$. This offset is the current reference's relative pad. It is determined (2) by a minimization function $f_{min}({}_gH, H, \text{offsets})$. The global histogram ${}_gH$ represents the conflict potential of the references padded so far. It is initialized with the histogram pertaining to special references like the stack area that cannot be padded easily. Each time a new offset has been determined, the global histogram is updated by merging it with the current histogram rolled by the amount of its offset. Frequently, there exists a set O_{min} of multiple offsets with a minimal or near minimal conflict potential according to f_{min}. These alternatives offer flexibility when optimizing simultaneously for further targets like,

for example, for efficient use of more than one level of a cache hierarchy.

Equipartition: A simple, intuitive approach is trying to make use of the whole cache. The function f_{min} in charge tries to achieve a uniform distribution of cache accesses by padding all references according to their accesses described in the *SRH* histograms. In each step of the algorithm, the current histogram H of a single reference and the current global histogram ${}_gH$ are merged with an offset found by a normal deviation guided function f_{min}. We first define the histogram ${}_jH$ as the global histogram after merging it with the current histogram H at offset j. Then, we define s_j as a measure for the deviation to be expected for offset j. The set S is defined as the set of values s_j for all available offsets j. The set O_{min} contains the offsets corresponding to a minimal or near minimal deviation. The parameter $\epsilon \geq 0$ controls the accepted range of results from which f_{min} chooses its result.

$$
\begin{aligned}
{}_jH &:= {}_gH \oplus (H \gg j) \\
s_j &:= \tfrac{1}{H_{size}} \textstyle\sum_{i=0}^{H_{size}-1} ({}_jH(i) - \tfrac{1}{H_{size}} |{}_jH|)^2 \\
S &:= \{s_j : j \in [0, H_{size})\} \\
O_{min} &:= \{i : s_i \leq \epsilon + min\ S\}
\end{aligned}
$$

This strategy is especially useful for applications whose memory footprint is smaller or not much larger than the cache size.

Inter-reference distance: Each non zero value in a PDH difference histogram points out a potential conflict for the pair of references concerned. The histogram bars weight the conflict distances between the two references. A single bar's height indicates the conflict potential for the bar's offset. Thus, in order to avoid mutual cache thrashing the difference in the padding offsets for the two references concerned, is chosen to correspond to a bar as low as possible considering that this optimization has to be done simultaneously for all PDH-histograms of the current reference with the references already padded in the global scope.

Self distance restrictions: In the cases of multiple similarly optimal offsets determined by other strategies, the histogram type *SDH* can additionally optimize for k-line prefetching. Prefetching is very susceptible to memory bandwidth, which can be lowered by padding references with similar self distances, as described by the SDHs, to cache regions avoiding continued thrashing.

After determining the offsets forming the runtime pad set, the memory allocation can be further optimized without influencing the padding offsets to minimize the total memory usage. This optimization is not in the scope of this paper. It resembles the enlarged backpack problem for a multiple count of packs.
Finally the optimization results are stored in a single file associated with the executables name.

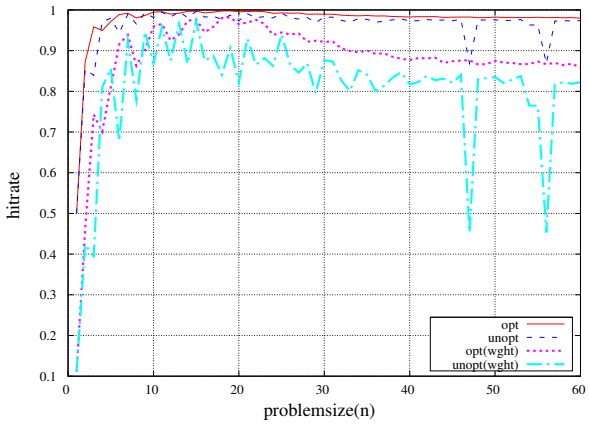


Figure 2: Cache Hit Rate for LU Decomposition on a 4k Direct Mapped Cache with Line Size 32

D. Optimized Execution

The final target of the considerations above is the optimized cache aware execution of the traced program. We achieve it with one of two options:

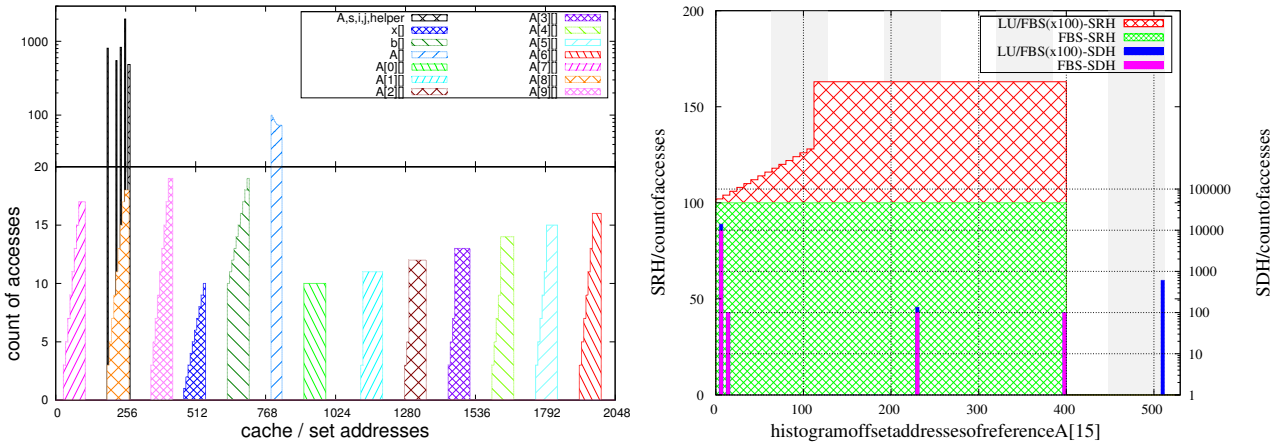
ld_preload: In order to intercept the original memory allocation operations associated to individual references, we have to catch and to patch all `malloc`, `free` and `realloc` system calls during the runtime of the program to be optimized with the help of the pre-determined pad set. A simple but effective way of doing this, is to manipulate the internal hooks of the memory management system. Therefore, a preloaded code overriding the original allocations was written, which sets up the pad offset of each dynamically allocated reference. The original program code does not need to be aware of these circumstances, in fact, the base address is shifted to the optimized offset but from the application's perspective it appears as a normal memory allocation.

Compiler, Datatypes: Having the source of the application to be optimized, the compiler itself is capable to shift the references to the pre-determined offsets, as well. In this case, all pads are already set and fixed up in the compilation. Besides, this source level patching offers zero runtime overhead for accessing the pre-determined padded structures by simple base address shifts which have to be done in the unoptimized case, too.

In our testing environment we actually do the padding by applying the pad set to the self protocoling data types.

IV. EXPERIMENTAL RESULTS

To show the usability of the concept of automated simulation driven padding, we chose the often used matrix data structure and, as a typically applied operation, the matrix multiplication. The multidimensional structures are made up of separate arrays in order to keep the opportunity for reordering and to maximize the number of references that could be padded. This assumption reflects the reality of often used row or column major data types in high level programming languages. C-programs should



(a) (b)

Figure 3: (a) Histogram for LU Decomposition with n=10 (b) SRH, SDH of the Row Reference A[15]

use row-major memory alignment for efficient memory access, but not all programs are optimized in this way. For this reason, non language conform data to memory mappings are interesting, too, for showing that padding can support a better cache performance for such programs, as well.

In the following, we present the results of examining and optimizing a few different applications.

A. LU decomposition to solve linear equation systems

We examine data intensive solvers as the *Gauß*-algorithm in order to design target oriented optimization heuristics. In Figure 3(a) the first histogram type *SRH* shows the access patterns of each reference with their original pads. There are graphs for the rows A_i of matrix A, its row pointers $A[]$, the vectors x, b and the stack segment including the local variables as well as loop iterators, accumulating variables and the pointer to the matrix A. The chosen problem size of $n = 10$ allows to conveniently display at once all the access patterns resulting from solving a linear system of equations.

In order to solve the problem $Ax^{(i)} = b^{(i)}$ several times with a modified vector $b^{(i)}$ and a static coefficient matrix $A \in R^{n,n}$ we can decompose A into a lower triangular matrix L and an upper triangular matrix U with $LU = A$. Both of the triangular matrices are stored in the original places of the former quadratic coefficient matrix A. U is formed by applying the Gauß elimination algorithm to A while L holds the corresponding elimination factors. The computation of L and U takes place only once and has an asymptotic complexity of $O(n^3)$. The forward substitution $Ly = b^{(i)}$ derives y from the current vector $b^{(i)}$ and the backward substitution $Ux^{(i)} = y$ solves the proper problem. Both operations are of complexity $O(n^2)$ and address the same references the LU decomposition already accessed.

Figure 3(b) on the right side shows the histograms *SRH* and *SDH* of the single row A_{15}'s accesses in one diagram for solving the problem $Ax^{(i)} = b^{(i)}$ of the size $n = 50$ for 100 different vectors $b^{(i)}$. The pattern for ${}_SH_{A_{15}}$ is as expected and mixes phases one and two by counting all accesses. The overlayed (right axis) histogram ${}_DH_{A_{15}}$, depicts the difference patterns split according to the two sequential phases of decomposition and solution. The pattern ${}_DH$ of a matrix row generates, in addition to the linear inner loop sequential accesses, spikes for reverse directed accesses for every line rewind enforced by the outer loop of the decomposition. The 100 forward and backward substitutions are computed in place by altering the elements of vector b. In the difference pattern histogram, they show a behavior similar to that of the LU decomposition.

Figure 2 compares the performances before and after optimization for different system sizes n. One measurement comprises one LU decomposition and 100 repeated solutions. The performance of the optimized versions is consistently higher and the results behave much more stable than with the original, straightforward padding provided by the memory allocation system.

B. Matrix-Matrix Multiplication

The results for another typical problem of an optimized single matrix-matrix multiplication $C = A \cdot B$ for different system sizes n shown in Figure 4 are padded into a 4k direct mapped cache with line size 32 and hit / miss latencies of $t_{L_H} = 1$ and $t_{L_M} = 7$, respectively. The matrices $A, B, C \in R^{n \times n}$ can be stored in row- or column-major order. For the experiment's results presented in Figure 4(a) a unique row-major ordering for all three matrices was chosen (row-row-row ordering). For small n, compulsory misses are dominant and memory bandwidth is poorly used with only partially accessed cache lines. For $n < 14$ the whole problem can be cache contained although the unoptimized allocation scheme arranges references' footprints usually cyclically within the cache issuing larger chunks of memory aligned to memory paragraphs, compare Figure 3(a). Our current padding heuristic compacts all references for minimal, but effective

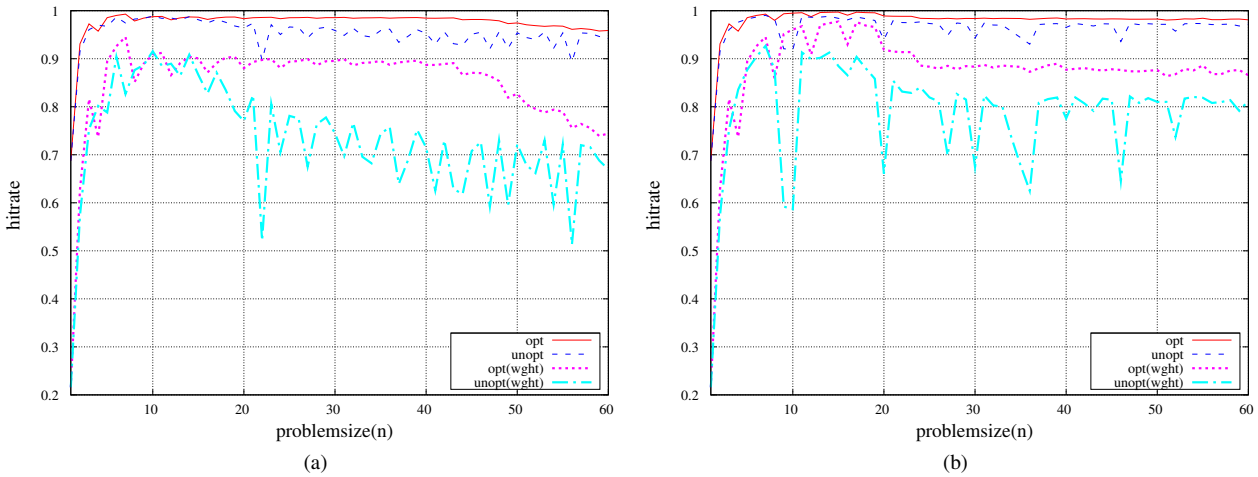


Figure 4: Matrix-Matrix Multiplication on a 4k Directed Mapped Cache with Line Size 32; (a) Exclusive Row, (b) Row-Column Ordered

cache usage. With increasing problem size, the original program runs suffer early of slight but continuous degeneration of cache performance because the probability of thrashing grows with n. Badly chosen memory alignments drastically impact the over-all hit rate quite often. The *weighted* plots in the figures discussed give an impression of the runtime effect based on the efficiency of the program's data cache usage. Given the numbers h of hits, m of misses and the latencies t_{L_M} and t_{L_H}, we estimate the total sum of cycles used to access memory as $h \cdot t_{L_H} + m \cdot t_{L_M}$. The weighted hit rate hr_w is then defined as follows:

$$hr_w = \frac{h \cdot t_{L_H}}{h \cdot t_{L_H} + m \cdot t_{L_M}}$$

Figure 4(b) depicts the results measured for a more efficient implementation with row-column-row major memory alignment of the three matrices A, B and C, respectively. Due to the smaller number of potentially conflicting cache lines, the over-all performance is further increased compared to the row-row-row major memory ordering in Figure 3(a).

C. Runge-Kutta solver for ODEs

As an example dealing with different data structures and access patterns, we examined an application to solve ordinary differential equations (ODEs) with a Runge-Kutta algorithm. For certain problem sizes this application displayed unexpectedly long run times. Using our toolset we were able to identify a thrashing constellation in these configurations. An analysis of the access patterns enabled us to devise a new solver with a consistently improved memory behavior [Korch and Rauber, 2006].

V. RELATED WORK

Techniques for increasing the locality of memory references have been studied extensively. An important, purely analytical model for effects of loop transformations [Abella et al., 2002] partly combined with padding and tiling can evaluate cache performance using CMEs (Cache Miss Equations) [Ghosh et al., 1997], [Vera et al., 2004] for special and zoned loop constructs. Another approach [Scholtes, 2003], based on conflict classes, does so for the Cholesky factorization. For applications with seemingly irregular or complex access patterns, an alternative memory mapping can be applied by evaluating indices with the help of an easy to compute map function. The memory access patterns generated by these functions are designed to have a better cache performance but they are strictly problem specific [Coleman and McKinley, 1995]. The Morton-Ordering is such a more general approach optimizing the memory layout for matrix operations as proposed in [Thiyagalingam and Kelly, 2006]. Many scientific libraries like LAPACK [Anderson et al., 1999] are based on the BLAS (Basic Linear Algebra Subprograms) that can be considered as a de facto standard for linear vector matrix based numerical algorithms. ATLAS (Automatically Tuned Linear Algebra Software) [Whaley et al., 2001] provides an efficient implementation of BLAS routines, as well as a genetic [Vera et al., 2003] CME guided algorithm for detecting well performing tile sizes [Jin et al., 2001], [Rivera and Tseng, 1999] and pad offsets [Vera et al., 2002].

VI. CONCLUSION

To our knowledge we present a new approach to design a post compilation performance optimization tool set for general use without necessarily having the source code of the application to be optimized available. Additional tools are provided for analyzing graphically the data acquired in the first phase trace run and the results of the optimizations. One of these is an extensible cache simulation engine with a graphical front end supporting the issues of our optimization strategies. It is also suited for guiding a manual program optimization.

The proposed automated optimization algorithm uses the data previously acquired and stored into histograms describing access patterns. It balances the potential

of single references to cause cache conflicts. Measured results show a significantly improved cache performance. Unfortunate memory allocations producing excessive thrashing are avoided altogether.

For future, more sophisticated optimization heuristics accounting for multi-level caches, the different memory hierarchy latencies will be used to refine the prediction of miss penalties and, along with this, the automatic optimization. Furthermore we expect it to be possible to extrapolate the memory behavior of an application from a few small problem sizes to larger problem sizes by spreading the measured histograms accordingly. Our distance histograms currently disregard the time passed between two accesses whose distance is recorded. We plan to introduce a temporal component reflecting this time in order to provide a better base for optimizations.

REFERENCES

[Abella et al., 2002] Abella, J., Gonzàles, A., Llosa, J., and Vera, X. (2002). Near-optimal loop tiling by means of cache miss equations and genetic algorithms. In *Proceedings of the 2002 ICPP Workshops*, pages 568–577.

[Anderson et al., 1999] Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarlin, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide, Third Edition*. Society for Industrial and Applied Mathematics.

[Coleman and McKinley, 1995] Coleman, S. and McKinley, K. S. (1995). Tile size selection using cache organization and data layout. In *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI'95), SIGPLAN Notices*, La Jolla, CA.

[Ghosh et al., 1997] Ghosh, S., Martonosi, M., and Malik, S. (1997). Cache miss equations: An analytical representation of cache misses. Workshop on Interaction between Compilers and Computer Architectures, Third International Symposium on High-Performance Architectures (HPCA-3).

[Handy, 1998] Handy, J. (1998). *the Cache Memory book*. Academic Press, Inc., 2nd edition. THE authoritative reference on cache design.

[Jin et al., 2001] Jin, G., Mellor-Crummey, J., and Fowler, R. (2001). Increasing temporal locality with skewing and recursive blocking. In *SC2001: High Performance Networking and Computing*. ACM Press and IEEE Computer Society Press. CD-ROM.

[Korch and Rauber, 2006] Korch, M. and Rauber, T. (2006). Optimizing locality and scalability of embedded Runge-Kutta solvers using block-based pipelining. *J. Par. Distr. Comp.*, 6(3):444–468.

[Larus, 1993] Larus, J. R. (1993). Efficient program tracing. In *Proceedings of IEEE Computer*, pages 52–61.

[Magnusson et al., 2002] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G., Hgberg, J., Larsson, F., Moestedt, A., and Werner, B. (2002). Simics: A full system simulation platform. *Computer*, 35(2):50–58.

[Magnusson et al., 1998] Magnusson, P. S., Dahlgren, F., Grahn, H., Karlsson, M., Larsson, F., Lundholm, F., Moestedt, A., Nilsson, J., Stenstrøm, P., and Werner, B. (1998). SimICS/sun4m: A virtual workstation. In *Usenix Annual Technical Conference*, New Orleans, Lousiana.

[Mauro and McDougall, 2001] Mauro, J. and McDougall, R. (2001). *SOLARIS Internals*. Sun Microsystems Press - A Prentice Hall Title.

[Oren, 2000] Oren, N. (2000). A survey of prefetching techniques. Technical Report number CS-2000-10, University of the Witwatersrand, Johannesburg, South Africa.

[Rivera and Tseng, 1999] Rivera, G. and Tseng, C.-W. (1999). A comparison of compiler tiling algorithms. In Proceedings of the 8th International Conference on Compiler Constuction (CC'99).

[Scholtes, 2003] Scholtes, C. (2003). *Abschätzung der Fehlzugriffe bei dünn besetzten Matrixoperationen auf Architekturen mit einem direkt mapped Cache*. Dissertation.

[SPARC, 2000] SPARC (2000). *The SPARC Architecture Manual Version 9*. SPARC International, Inc.

[Thiyagalingam and Kelly, 2006] Thiyagalingam, J. and Kelly, P. H. J. (2006). Is morton layout competitive for large two-dimensional arrays ? *Concurr. Comput. : Pract. Exper.*, 18(11):1509–1539.

[Vera et al., 2003] Vera, X., Abella, J., Gonzalez, A., and Llosa, J. (2003). Optimizing program locality through cmes and gas. *pact*, 00:68.

[Vera et al., 2004] Vera, X., Bermudo, N., Llosa, J., and Gonzalez, A. (2004). A fast and accurate framework to analyze and optimize cache memory behavior. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(2):263–300.

[Vera et al., 2002] Vera, X., Llosa, J., and González, A. (2002). Near-optimal padding for removing conflict misses. In Pugh, W. and Tseng, C.-W., editors, *LCPC*, volume 2481 of *Lecture Notes in Computer Science*, pages 329–343. Springer.

[Whaley et al., 2001] Whaley, R. C., Petitet, A., and Dongarra, J. J. (2001). Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35.

Hardware Design, Expandability, System Cost and Mean Inter-node Message Distance of Augmented Hypercube Torus and Master-Slave Star-Ring Augmented Hypercube Architectures

Maryam Amiripour, Hamid Abachi
Dept. of Electrical and Computer Systems Engineering
Monash University
Australia
Maryam.Amiripour@eng.monash.edu.au
Hamid.Abachi@eng.monash.edu.au

Abstract— **The need for high computation power, as well as advances in the communications technology have resulted in the rapid development of high-performance message passing architectures. Scalability and cost issues which are part of the performance evaluation in parallel processing systems are recognized to be challenging tasks and are considered as the main measures to identify the suitability of the topology for a given application.**

In this paper the most common message passing architectures together with the Augmented Hypercube Torus (AHT) are described and compared with a newly developed architecture called Master-Slave Star-Ring Augmented Hypercube. Its expandability, hardware cost and Mean Inter-node Message Distance (MIMD) are evaluated for various network sizes and their merits and demerits are highlighted.

Keywords— **Expandability, Routing, Reliable Parallel and Distributed Systems, Performance, Message Passing.**

I. Introduction

Although tightly-coupled systems can provide cost effective improvement on the computing power of a single processor, due to their nature, they may suffer from serious bus contention when global shared memory is used. In an attempt to overcome the limitations of memory contention and rather poor performance associated with shared memory architectures, message-passing systems were introduced. These architectures include Torus, Hypercube and Tree systems.

These message passing systems are mainly used in multi-dimensional configurations. In these topologies, processors instead of having access to a common memory, have their own local memory and communication links to other processors to share information, thereby greatly reducing contention [1].

In general, Hypercube architecture can be expanded by increasing its dimensionality (h). Expanding a Hypercube causes an increase in dimensionality which requires more ports per processor. In general, the maximum number of nodes is limited by the fixed number of processor ports. If each node (N) in a Hypercube architectures is a traditional processor, then it can only communicate with one processor at a time (e.g. over a common bus) [2], [3]. Consequently performance is reduced due to lack of simultaneous communication capability with other nodes. One solution to overcome this issue is to implement the SGI Altix NUMA flex architecture. This product uses an SGI NUMA (cache coherent non-uniform memory access) protocol implemented directly in hardware for performance and a modular packaging scheme. The key to the NUMA flex design of Altix is to use a controller ASIC, referred to as the super Hub (SHub) that interfaces to the Titanium 2 front side bus, together with the memory as well as the I/O subsystem, which further interfaces with other NUMAflex components in the system [4]. This provides simultaneous communication between processors in a true message passing environment. We can implement the proposed architecture by using CR brick which houses 4 NUMA flex nodes totalling 8 Intel Itanium 2 processors. Each NUMA flex node has 12 slots and currently supports 2 GB memory [4]. The CR-brick architecture satisfies our satellite node (slave processor) configuration requirements, which simply means it has eight processors including crossbar switches as routers, so that we can consider each slave component to be equivalent to one CR-brick module.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

II. A Torus of Augmented Hypercube

Compared to the Hypercube, Torus and Tree networks are infinitely expandable by increasing w (width) or n (levels) and keeping t (dimension) or b (branches) constant. No network re-wiring is needed for the tree when nodes are added to the last level, because nodes are appended onto the unconnected branches of the Tree. For a Torus, only minor network re-wiring is required when nodes are added, because nodes need to be inserted into the network [5].

By connecting Augmented Hypercubes (AHs) in a Torus through Routers, an infinitely expandable network is possible by increasing the torus width. It is also possible to have a Tree of AHs, or to replace the AH with any other "augmented" structure, and substituting these with the nodes in some other structure [6]. For the purposes of comparison in this paper, we will limit our discussion to the AHT, Hypercube, Torus and Tree architectures together with a newly proposed architecture called MSSRAH. Figure 1 illustrates a typical AHT architecture.

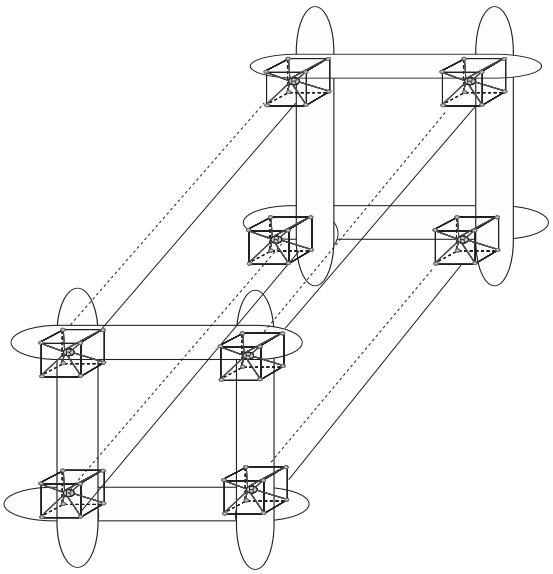


Fig. 1: Augmented Hypercube Torus (AHT) Architecture.

III. Newly Proposed Architecture MSSRAH

We propose a newly developed architecture called Master Slave Star Ring Augmented Hypercube (MSSRAH) configuration. The master processor in this configuration is at the center of the ring and can provide access to each satellite node through fast and reliable communication links. The structure of satellite nodes and the master processor is basically the same although the master processor is faster and has more memory capability plus other supporting hardware and software tools that normally is the requirement for such high speed architecture. This configuration is depicted in Figure 2.

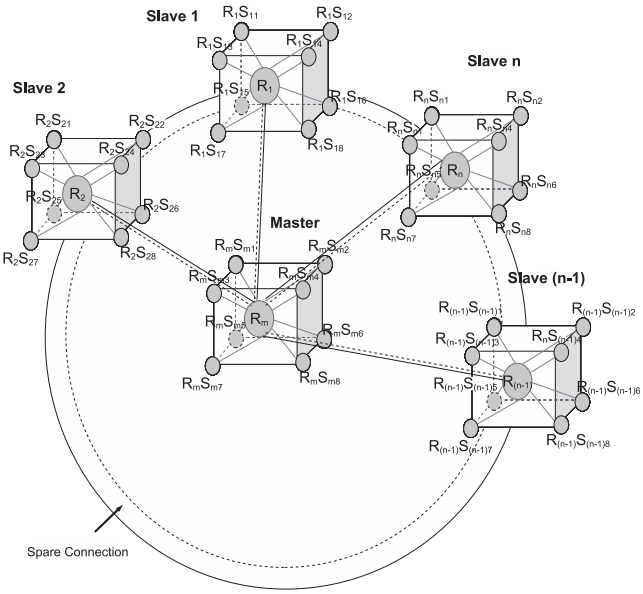


Fig. 2: The Overall Configuration of the Master-Slave Star-Ring Augmented-Hypercube Architecture.

IV. Network Modeling

For a meaningful comparison among popular existing parallel processing architectures with the new ones proposed here, Table 1 summarises parameters of Torus, Hypercube, Tree, AHT and MSSRAH network topologies.

It is straightforward to derive equations for these parameters in the case of the Torus, Hypercube and Tree and AHT networks [7].

The MSSRAH is a star of AHs so that the total number of processing nodes (N_N) is given as the product of the number of Hypercube nodes and the number of Star nodes. This results in:

$$N_N = 2^h \times n \tag{1}$$

where n is the number of Star nodes and 2^h is the number of Hypercube nodes.

The number of communication links (N_L) for the MSSRAH is complicated by the presence of the Router. We will partition the MSSRAH into a Star, Ring and AHs.

TABLE I: Parameters of Torus, Hypercube, Tree, AHT and MSSRAH network topologies.

Architecture	**Number of Nodes(N_N)**	**Number of Links(N_L)**
Torus,t=dim,w=width	w^t	tw^t
Hypercube, h=dim.	2^h	$h2^{h-1}$
Tree,b=branches,n=levels	$\frac{b^n-1}{b-1}$	$\frac{b^n-b}{b-1}$
AHT, t=Torus, w=Torus width, h= Hypercube dim.	$2^h w^t$	$(h2^{h-1}+2^h)w^t + tw^t$
MSSSRAH, n= Number of Star nodes, h= Hypercube dim.	$2^h n$	$(h2^{h-1}+2^h)n + 2(n-1)$

In the MSSRAH there are n AHs, and the number of links in the Star configuration is $n-1$. Since we have a Star topology with n nodes, therefore, the number of links in the Ring configuration is also $(n-1)$. If each AH has N_{LAH} links, then the total number of links is:

$$N_L = 2 \times (n-1) + n \times N_{LAH} \tag{2}$$

Within an AH section there are links directly connected to the processing nodes (in a Hypercube topology), and from each AH node runs a link to the Router. Thus the number of links for a AH segment is:

$$N_{LAH} = h \times 2^{h-1} + 2^h \tag{3}$$

This gives an expression for the number of links for the entire MSSRAH topology, which is dependent on the number of star nodes, and AH dimensionality. This results in:

$$N_L = (h2^{h-1} + 2^h) \times n + 2(n-1) \tag{4}$$

Once the number of processing nodes and the number of links are defined, then the hardware cost analysis can be made as follows.

V. Hardware Cost Analysis

In the context of parallel processing systems, cost is a difficult parameter to define, especially given that component costs are highly dependent on implementation and economic conditions. In general, overall total system cost estimate (C_{ST}) as reported in [8], can be shown as:

$$C_{ST} = C_N(N_N + K_R N_R) + C_L \times N_L \tag{5}$$

where, $K_R = \frac{C_R}{C_N}$, and
- C_N= Cost of node
- C_L= Cost of link
- N_R= Number of Routers
- C_R= Cost of Router

However, a far more difficult question for the multiprocessor system designer is "How well suited is a network over a range of component costs?." In particular one can examine the total system cost when it is compared to the total processing node cost. This can be done since such a figure describes how close a particular network is to the ideal lowest cost network where there are no Router or communication link overhead costs (i.e. $C_R = 0$ and $C_L = 0$, giving $C_{ST} = C_N N_N$) [3]. Thus one can normalise the total system cost function C_{ST}, by $C_N N_N$ giving

$$K_{ST} = \frac{C_{ST}}{C_N N_N} = \frac{K_R N_R + K_L N_L}{N_N} \tag{6}$$

where, $K_L = \frac{C_L}{C_N}$. The normalised total system cost, K_{ST}, then gives us the total system cost relative to the lowest theoretical system cost. The results are summarised in Table 2. It should be noted that only the AHT has a K_R parameter, as it is the only system utilising Routers. For other systems we set $N_R = 0$ as they do not use Routers.

In practice K_L will vary from near zero for a tightly coupled multi-processor system to less than one for a distributed computer network. For tightly and closely coupled multi-processor systems, it has been suggested that $K_L = 0.1$ is a reasonable value [4]. Based on the result of N_L for MSSRAH shown in (4), the K_{ST} can be formulated as follows:

$$K_{ST} = 1 + \frac{K_R N_R + K_L N_L}{2^h n} \tag{7}$$

Since the Number of $N_R = n$ (number of Star nodes), then:

$$K_{ST} = 1 + \frac{K_R}{2^h} + K_L \frac{[(h2^{h-1} + 2^h)n + 2(n-1)]}{2^h n} \tag{8}$$

or,

$$K_{ST} = 1 + K_R 2^{-h} + K_L \frac{[(h2^{h-1}n + 2^h n) + 2n - 2)]}{2^h n} \tag{9}$$

After further simplification and rearrangement, K_{ST} can be expressed as:

$$K_{ST} = 1 + K_R 2^{-h} + K_L [\frac{h}{2} + 1 + \frac{n-1}{2^{h-1}n}] \tag{10}$$

TABLE II: Normalised system cost K_{ST} for Torus, Hypercube, Tree, AHT and MSSRAH architectures.

Parameter	**Torus**	**Hypercube**	**Tree**	**AHT**	**MSSRAH**
K_{ST}	$1+K_L t$	$1+K_L\frac{h}{2}$	$1+K_L\frac{b^n-b}{b^n-1}$	$1+K_L[2^{-h}t+\frac{h}{2}+1]+K_R 2^{-h}$	$1+K_L[1+\frac{h}{2}+\frac{n-1}{n2^{h-1}}]+K_R 2^{-h}$

The overall results for K_{ST} for the message passing architectures including newly proposed architecture are summarised in Table 2.

VI. Significant of K_R for the AHT/ MSSRAH Architecture

Due to the structure of the AHT, the number of Routers is small compared to the number of processing nodes. This may not be true of other structures. It will be demonstrated that for AHT structures of interest in this paper ($h \geq 3$) the cost of Routers is insignificant in comparison to the total system cost [9].

The normalised total system cost for the worst case occurs where links are very cheap ($K_L = 0$). Even with a very pessimistic estimate namely $K_R = 1$ which means the Router's cost is the same as the processing node cost (a realistic system would have a much smaller value, given that processing nodes are far more complex and involve more sub-systems) for a three dimensional AH ($h = 3$), the Routers contribute only %12 to the total system cost. Clearly, from the above formula, the Routers contribute exponentially less as the AH dimensionality increases. Thus, one can ignore the Router's cost contribution and assume $K_R = 0$, whereas the K_{ST} limit for infinite size of MSSRAH would be:

$$1 + K_L[\frac{h}{2} + 1 + \frac{1}{2^{h-1}}] \qquad (11)$$

The graphical presentations of The Normalised System Cost versus the Number of Processing Elements are shown in Figures 3, 4 and 5.

The normalized cost K_{ST} gives us the ratio of the actual total cost to the ideal minimum system cost. In general, communication cost and consequently the system cost increase with increasing K_L. As can be seen in Figures 3, 4 and 5, Torus, AHT and MSSRAH have a constant K_{ST} for different values of h for ($h = 3, 4$ and 6) with $K_L = 0.1$. This implies that communication link costs are always a fraction of the processor costs. Hypercubes differ in this respect because K_{ST} increases as the number of processor nodes increases. This is undesirable since an increasing proportion of the system cost is devoted to communication network overheads and not processors.

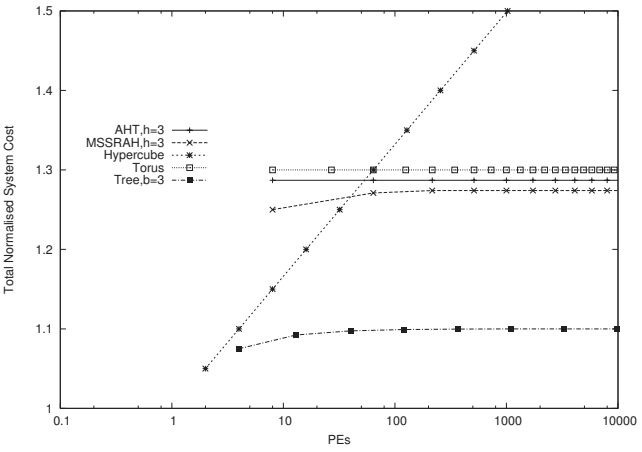


Fig. 3: Normalised System Cost for AHT, MSSRAH, Torus and Tree Networks with h=3 and K_L=0.1.

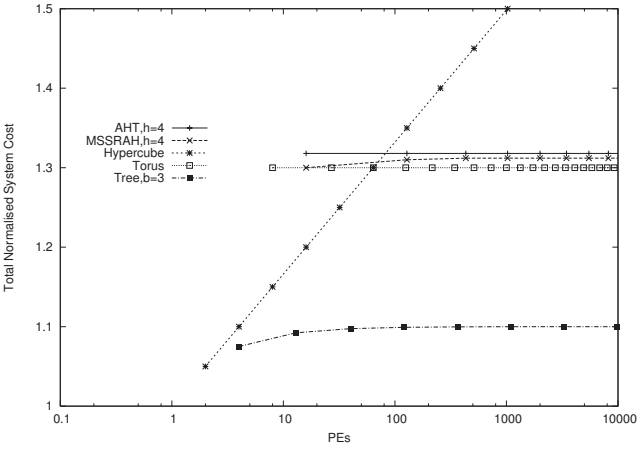


Fig. 4: Normalised System Cost for AHT, MSSRAH, Torus and Tree Networks with h=4 and K_L=0.1.

VII. Mean Inter-node Message Distance (MIMD) Analysis of AHT and MSSRAH Architectures

A. MIMD Analysis of AHT topology

As it is reported in [7], the Mean Inter-node Message Distance (MIMD) under uniformly distributed mes-

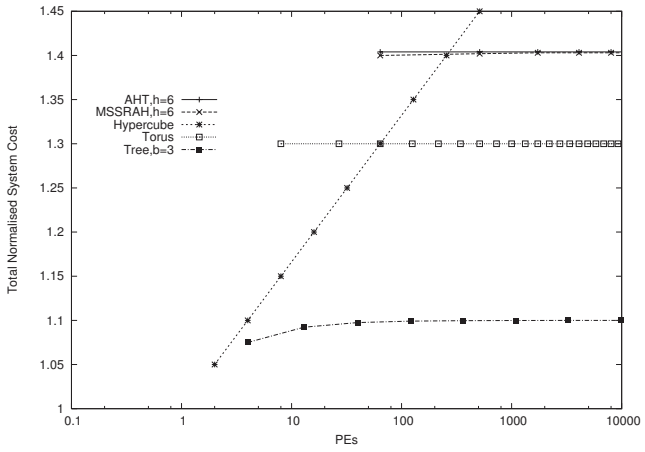


Fig. 5: Normalised System Cost for AHT, MSSRAH, Torus and Tree Networks with h=6 and K_L=0.1.

sages for AHT can be summerised as:

$$MIMD_{AHT} = \frac{tw - tw^{-1} + 8}{4} - \frac{h}{w^t(2^h - 1)}, (w - odd) \quad (12)$$

and,

$$MIMD_{AHT} = \frac{tw + 8}{4} - \frac{h}{w^t(2^h - 1)}, (w - even) \quad (13)$$

B. MIMD Analysis of MSSRAH topology

To find out the average routing distance of MSSRAH, a probabilistic view is taken to identify what fraction of the messages will route within a single AH node, and what fraction of messages will route to different AH nodes.
In this way, the sperate MIMD of AH and the 2-Level Tree can be combined to yield the MIMD of the MSSRAH.
To find the probability that the source (s) and destination (d) nodes are in the same AH (satellite slave) in MSSRAH, we consider the probability of event d, a destination AH node, given that event s, a source AH node has been chosen. The probability that any one particular node is chosen out of n nodes is:
p(s)=p(d)=$\frac{1}{n}$
Based on this assumption, it can be shown as:
$p_{sameAH} = p(d \mid s) = \frac{p(d \cap s)}{p(s)} = \frac{p(d)p(s)}{p(s)}$.
However, since source and destination nodes are independent, therefore:
$p_{SameAH} = p(d) = \frac{1}{n}$
According to [10],

$$MIMD_{AH} = 2 - (\frac{h}{(2^h - 1)})$$

For calculation of Mean Inter-node Message Distance of MSSRAH topology, we consider MSSRAH as a 2-level Tree which its nodes in level 2 have been connected together as a ring. Since all slave Routers have been connected together in pairs, therefore, MIMD for MSSRAH can be shown as:

$MIMD_{MSSRAH} = (MIMD_{2L-Tree_{b=n-1}} + 2) \times (\frac{n-1}{n}) + (MIMD_{AH}) \times (\frac{1}{n})$.
Thus in the general case we have:

$MIMD_{MSSRAH} = [(\frac{2b}{b+1}) + 2] \times (\frac{n-1}{n}) + (2 - \frac{h}{2^h - 1}) \times (\frac{1}{n})$.
After further simplification, rearrangement and considering $b = n - 1$, $MIMD_{MSSRAH}$ can be shown as:

$$MIMD_{MSSRAH} = [\frac{2}{n^2} - \frac{6}{n} + 4] + [2 - \frac{h}{2^h - 1}](\frac{1}{n}) \quad (14)$$

Table 3 provides a summary of different network metrics and Figure 6 illustrates the Mean Inter-node Message Distance for Torus, Tree, Hypercube, AHT and MSSRAH networks.

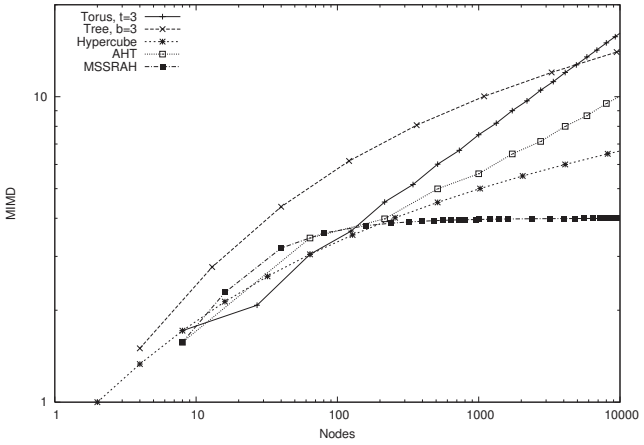


Fig. 6: MIMD for Torus, Tree, Hypercube, AHT and MSSRAH networks.

VIII. Visit Ratio

To calculate the Visit Ratio, we need to know the number of communication links in the network. As reported in [11], the Visit Ratio can be expressed as:

$$VR = \frac{MIMD}{N_L} \quad (15)$$

A. Visit Ratio of AHT topology

As reported in [11],the Visit Ratio of AHT can be shown as:

$$VR_{AHT} = \frac{\frac{tw - tw^{-1} + 8}{4} - \frac{h}{w^t(2^h - 1)}}{(h2^{h-1} + 2^h)w^t + tw^t}, (w - odd) \quad (16)$$

TABLE III: Summary of network metrics

Architecture	**Torus**	**Hypercube**	**Tree**	**AHT**	**MSSRAH**
N_N	w^t	2^h	$\frac{b^n-1}{b-1}$	$2^h w^t$	$2^h n$
diameter	$\frac{tw}{2}$	h	$2(n-1)$	$\frac{tw}{2}+2$	$\frac{n-1}{2}+2$
MIMD	$\frac{tw^{t-1}(w^2-1)}{4(w^t-1)}$ (w-odd) $\frac{tw^{t+1}}{4(w^t-1)}$, (w-even)	$\frac{h2^{h-1}}{2^h-1}$	$\frac{2nb^{n-1}(b^n+1)}{(b^n-1)(b^{n-1}-1)} - \frac{2b^{n-1}(b+1)}{(b^{n-1}-1)(b-1)}$	$\frac{tw-tw^{-1}+8}{4} - \frac{h}{w^t(2^h-1)}$, (w-odd) $\frac{tw+8}{4} - \frac{h}{w^t(2^h-1)}$, (w-even)	$[\frac{2}{n^2}-\frac{6}{n}+4]+[2-\frac{h}{2^h-1}](\frac{1}{n})$
N_L	tw^t	$h2^{h-1}$	$\frac{b^n-1}{b-1}$	$(h2^{h-1}+2^h)w^t+tw^t$	$(h2^{h-1}+2^h)n+2(n-1)$
VR	$\frac{w^2-1}{4w(w^t-1)}$, (w-odd) $\frac{w}{4(w^t-1)}$, (w-even)	$\frac{1}{2^h-13}$	$\frac{2b^{n-2}(b-1)}{b^n-1}$	$\frac{\frac{tw-tw^{-1}+8}{4}-\frac{h}{w^t(2^h-1)}}{(h2^{h-1}+2^h)w^t+tw^t}$, (w- odd) $\frac{\frac{tw+8}{4}-\frac{h}{w^t(2^h-1)}}{(h2^{h-1}+2^h)w^t+tw^t}$, (w-even)	$\frac{[\frac{2}{n^2}-\frac{6}{n}+4]+[2-\frac{h}{2^h-1}](\frac{1}{n})}{[(h2^{h-1}+2^h)n+2(n-1)]n}$

and,

$$VR_{AHT} = \frac{\frac{tw+8}{4} - \frac{h}{w^t(2^h-1)}}{(h2^{h-1}+2^h)w^t + tw^t}, (w-even) \quad (17)$$

B. Visit Ratio of MSSRAH topology

Considering the equation (15), the Visit Ratio of MSSRAH architecture can be evaluated as:

$$VR_{MSSRAH} = \frac{[\frac{2}{n^2} - \frac{6}{n} + 4] + [2 - \frac{h}{2^h-1}](\frac{1}{n})}{[(h2^{h-1}+2^h)n + 2(n-1)]n} \quad (18)$$

IX. Conclusion

This paper examines and compares through mathematical modeling and simulations, the expandability, hardware cost and MIMD analysis of a newly proposed architecture (MSSRAH) with the existing message passing architectures including the AHT, Hypercube, Tree and Torus architectures.

The MSSRAH architecture has a better scalability than the remaining message passing architectures. This simply indicate that it grows with consistency without loss of relative performance. This is evident by having the lower graph for MSSRAH architecture in Figure 6. The MSSRAH architecture not only performs better in terms of the relative cost of other message passing architectures mentioned above, but it also significantly improves the communication performance due to the existence of the Routers and provision of spare Ring link that ensures the system is rarely subject to catastrophic failure.

The distance between the markers on each curve in Figure 6 gives a good indication of the network granularity. Ideally we want the smallest granularity so that the network can be expanded at conveniently small increments.

Careful examination of this feature reveals that the Hypercube has evenly spaced markers which are far apart. This indicates a particular weakness of Hypercube. On the other hand Torus has a desired network granularity. AHT and MSSRAH topologies have an excellent degree of freedom over network granularity in particular MSSRAH, since it has lower curve which is indicative of better performance.

It is also expected that the latter architectures due to the existence of Routers in their centers offer some redundancy in communication so they may become useful in mission critical systems. These topologies require processors with a few communication ports that are supported by SGI technology which would offer performance enhancement.

References

[1] Kodase. S, Wang. S, Gu. Z and Shin. K.G, "Improving scalability of Task Allocation and Scheduling in Large Distributed Real-Time Systems Using Shared Buffers" *Proceedings of the 9th Real-time/Embedded Technology and Applications Symposium (RTAS), IEEE, Washington DC, U.S.A, 2003.*

[2] Walker. J. 1998, " Performance, Reliability and Cost Analysis of Message Passing Architecture" *Master of Engineering Thesis, Department of Electrical and Computer System Engineering, Monash University.*

[3] Bajaj. R, Agrawall. D.J, "Improving Scheduling of Tasks in a Hetergenous Environment" *IEEE Transactions on Parallel and Distributed Systems, Vol 15,No 2,Feb.2004,pp. 107-117.*

[4] Sillicon Graphics Inc. 2004, *Hardware: End-User, Altix 3700 Bx2*, System Overview, Chapter 3, U.S.A, pp.1-6.

[5] Zhuang. X, Libratore. V, "A Recursion-Based Broadcast Paradigm in Wormwhole Routed Networks" *IEEE Transactions on Parallel and Distributed Systems,Vol 16, No 11, Nov. 2005, pp. 1034-1052.*

[6] Abachi. H, Walker. J and Debnath. N. 2000, "Methods for Comparing the Reliability of Advanced Distributed Computer Networks" *International Conference on Computer Applications in Industry and Enginnering, U.S.A, IJCA, pp. 307-310.*

[7] Abachi. H and Walker. J. 1998, "Design and Performance Analysis of Superhypercube, Transputer Tours" *International Journal of Computers and their Applications (ISCA),U.S.A, pp. 1-10.*

[8] Abachi. H and Walker. J. 1996, "Network Expandability and Cost Analysis of Tours, Hypercube and Tree Microproces-

sor Systems" *28th IEEE Southeastern Conference on System Theory, Louisiana, U.S.A, pp. 426-430.*

[9] Abachi. H and Walker. J. 1995, "Scalability and Hardware Cost Analysis of Augmented Hypercube Tours Architecture" *International Conference on Computer Applications in Enginnering and Medicine, Indiana, U.S.A, pp. 232-237.*

[10] Amiripour. M, Abachi. H and Dabke. K. 2007, "Hardware Design, Cost and Diameter Analysis of Super Hypercube Array, Master-Slave Star- Ring Super Hypercube and Master-Slave Super-Super Hypercube 4-Cube architectures" *WSEAS Transactions on Computer Research, to be appeared in 2007.*

[11] Amiripour. M, Abachi. H and Dabke. K. 2007, "Hardware Cost Analysis of Master-Slave Star- Ring Super Hypercube and Master-Slave Super-Super Hypercube 4-Cube Architectures" *6th WSEAS International Conference on Software Engineering Parallel and Distributed Systems (SEPADS'07), Corfu Island, Greece, accepted and to be presented in Feb. 2007.*

A Peer-to-Peer Simulation Architecture

Bernardt Duvenhage and Willem H. le Roux

Council for Scientific and Industrial Research
Pretoria, South Africa
Email: bduvenhage,whleroux@csir.co.za

Abstract— **A distributed parallel and soft real-time simulation architecture is presented. It employs a publish-subscribe communication framework layered on a peer-to-peer Transport Control Protocol-based message passing architecture. Mechanisms for efficient implementation and control of information flow between simulated entities form part of the architecture. A light-weight base simulation object model is also employed to provide maximum modularity and extensibility while keeping complexity manageable. The simulation architecture evolved over time to allow for the efficient implementation of a system of systems, virtual simulation. It has been successfully applied in an air defence simulation as a decision support tool and for standard operating procedure concept evaluation.**

KEYWORDS

Distributed, Parallel, Soft Real-Time, Simulation, Architectures, Peer-to-peer, Publish-Subscribe.

I. INTRODUCTION

In a decision support environment, system of systems level simulations are applied to provide end-users with the capabilities to identify, define, implement (virtual) and evaluate concepts that would otherwise be costly, time-consuming or impractical. Systems of systems level simulators typically involve multiple modelled entities with complex interactions and are executed in virtual or constructive simulation modes [1].

This paper presents a simulation architecture that evolved over time during decision support to an air defence procurement programme [2, 3]. Although there was no need for a generic, re-usable architecture – It was to be only used for a single simulation environment – it still had to provide standardised interfaces for efficient integration of models, services and other simulation-logistical functions. It should also support both constructive and virtual simulations, hence it should at least be soft real-time compatible when performing operator-in-the-loop simulations. Operators will mainly interact with the simulation via integrated mock-up consoles and not full immersive synthetic environments, which may be supported by integrated, external systems. Furthermore, it must allow both distributed and non-distributed simulation execution. The first is required to maintain soft real-time compliance when employing the virtual simulation mode if model processing loads are high. The latter is required for easier test and debugging as well as batch executions for statistical analyses. A conservative, discrete stepped time management mode forms an inherent part of the simulation architecture, as almost all of the models used in the air defence simulation environment are discrete time-stepped. To provide an efficient and effective decision support capability, specifically during system conceptualisation and field exercises, the architecture should allow for quick implementation and integration of new models. The same holds for the integration of external systems. Interoperability with other simulations is not an absolute requirement, but should not be excluded by design.

Several peer-to-peer architectures are reported in the literature, of which some are aimed at internet-based information sharing, discrete event simulations [4]–[7] or cooperative computing, such as solving processing intensive problems with *ad hoc* peer-to-peer networks [8]. Some architectures are also aimed at massively multi-player online role playing and other games [9, 10]. Giesecke [11] quantitatively investigated availability in peer-to-peer systems for prediction and identified basic characteristics to derive a formal model for describing architectures. Kotilainen [12] reports on an efficient peer-to-peer network simulator used to study artificial neural network algorithms.

Other simulation architectures or frameworks include the Aggregate Level Simulation Protocol (ALSP) [13], Distributed Interactive Systems (DIS) [14] and the High Level Architecture (HLA) [15]. The first two are seen as precursors to HLA. Although all three were developed for the defence community of the United State of America, HLA was intended to be adopted by the wider simulation community. The Standard Simulation Architecture [16] provides an additional framework to HLA to allow more flexibility, but be more cost-effective without paying performance penalties. The Open Simulation Architecture (OSA) [17] is a discrete event simulation architecture that promises integration of new and existing contributions at all levels. Hawley [18] proposes an object-oriented simulation architecture that separates the implementation of the dynamic system being modelled (application layer) from the simulation management functions (executive layer). The Extensible Modelling and Simulation Framework (XMSF) [19] aims to harness web-based technologies to promote interoperable simulations and provides mechanisms for systems to discover and use web services.

Of these architectures only HLA was evaluated since it is a fully fledged approach that covers all aspects of the simulation life cycle. However, in the South African defence environment it was not the optimal choice at the time. Although HLA promotes interoperability, the federation object model should still

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

be agreed or translated when two simulations are integrated. This was not always the case, therefore interoperability was not easily achievable [3].

The simulation architecture presented is not offered as an alternative for the above-mentioned architectures or frameworks, but rather to highlight the mechanisms used to implement an efficient architecture against the backdrop of system of systems simulation criteria. Efficiency in terms of implementation is required since a very small development team was used. In terms of simulation execution, soft real-time execution for multiple entities with update rates of 100Hz are used, requiring an architecture with low overheads.

II. System of Systems-level Simulation Architecture Needs

This section addresses the specific needs for a simulation architecture to meet the criteria as outlined in Section I and is discussed in the following subsections.

A. System of Systems Simulation

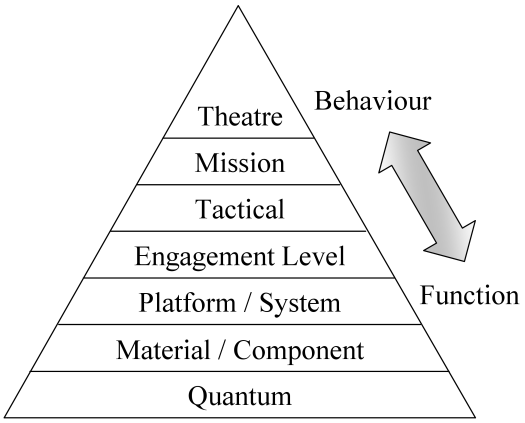


Fig. 1: The Systems Hierarchy applied to Modelling and Simulation (Adapted from [1, 20])

An adapted version of the systems hierarchy is shown in Figure 1 as applied to modelling and simulation [1, 20]. Systems of systems simulations are typically applicable at the engagement and tactical levels where entities are modelled at equipment and individual operator level. Individual entities are modelled, but forms "systems" when grouped organisationally. Tactical simulations also require one-on-one engagements, whereas higher order simulations require strategic actions of aggregated entities. For mission and theatre simulations entities are aggregated into units and generally referred to as war-gaming simulations [1]. Lower system hierarchy simulations tend to focus more on the function of entities, whereas the higher levels on the behaviour of entities.

B. Constructive and Virtual Simulation Mode Support

Tactical level simulations (Figure 1) imply that not only equipment is simulated, but also the use of equipment, including standard operating procedures. This again implies that the human operators, human-equipment and human-human interactions be modelled, so that it becomes a constructive simulation. Simulation execution requirements for constructive simulations tend to be less stringent, but the faster a simulation executes, the more applicable it becomes as a what-if type analyses tool, as it allows a simulation user to test and evaluate scenarios quickly. Virtual simulations need to be at least soft real-time compliant to maintain realism [21].

C. Distributed and Non-Distributed Simulation

Non-distributed simulations are generally less complex to test and debug than distributed simulations, as simulation execution does not have to be traced across multiple processing nodes. However, non-distributed simulations may be soft real-time incompatible when model processing loads become too high for a single processing node.

Distributed simulations on the other hand require efficient inter-process communication frameworks, such that the inter-processing node communication overheads do not counter the advantage of extra processing nodes. Virtual simulations with multiple entities that are modelled at system of systems level, typically require distributed simulation to either provide faster than or real-time compatibility. The ideal simulation architecture would support both distributed and non-distributed simulation execution without having to alter the implementation, but only its configuration.

D. Modularity and Extensibility

Since the simulation architecture is used in a decision support environment, including the evaluation of system concepts, it should be efficient to add, maintain and upgrade models of equipment, operator terminals, external system interfaces and operators. In addition to the entities that participate in a synthetic environment, it should also be efficient to extend, maintain and upgrade the synthetic environment itself. Services such as inter-entity line of sight calculations should be inherently part of the synthetic environment. External system interfacing should be supported, but note that external systems may have requirements that cannot be met by the simulation architecture, such as hard real-time compatibility.

E. Time-stepped Simulation

Conservative time management is an integral part of the simulation architecture and is enforced by using a discrete time-stepped mechanism. Spatio-temporal properties play a pivotal role in any air defence system, therefore time-line accuracy is of importance in a simulation environment, and hence architecture.

Although models may internally use predictive event-based time management, their external interfaces should support conservative time management. This is necessary as both predictable and non-predictable events occur in an air defence simulation environment, of which the non-predictable events may violate causality, if non-conservative time management is used.

Most of the external systems that will be integrated, produce spatio-temporal data, be it in the form of positions of an

aircraft from a flight simulator, or time-stamped detections from a sensor such as a radar. External systems that may be integrated includes equipment, data sources and simulators.

III. High-Level Simulation Architecture Design

In order to meet the needs as identified in Section II, a simulation architecture evolved from a single application to a fully distributed simulation architecture. After providing a short overview of the present architecture, each part is carefully explained in subsequent subsections.

The simulation architecture is based on an inter-process communication (IPC) framework using the Transfer Control Protocol (TCP). Processing nodes are fully connected in a peer-to-peer fashion and message-passing is managed via a publish-subscribe mechanism. Processes that need to communicate within the simulation architecture are:

- Models - Models of equipment, humans and operator consoles (interfaces).
- Services - Includes line-of-sight, terrain elevation and peripheral services such as data loggers.
- Consoles or Gateways - All external systems that need to be integrated with the simulation architecture is implemented via a gateway which in effect translates the protocol of the external system into the simulation object and spatial reference models of the simulation architecture. Mock-up operator consoles are also integrated via this mechanism.

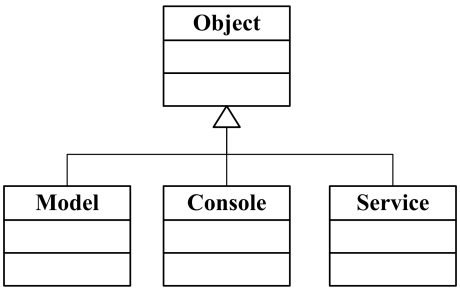


Fig. 2: Base Simulation Object Model

The above list of items is grouped under a base object to form the simulation object model of the architecture as indicated in Figure 2. An economical simulation object model (SOM) has been designed to curb implementation complexity.

A. Publish-Subscribe Object Communication Framework

All models, services and consoles (hereafter collectively referred to as objects) should be able to communicate efficiently in a distributed environment. Furthermore, it should be easy to establish and manage the communication channels between objects. A layered approach will reduce future migration effort to other IPC frameworks: keep the object implementation and communication logistics of an object separate. The object communication framework should also hide the underlying distributed implementation from the user of the framework and not dictate model fidelity or simulation granularity. The framework should allow distributed and non-distributed execution for easier test and debugging without requiring implementation changes of the framework or client software.

The publish-subscribe mechanism employed in the object communication framework is analogous to magazine subscriptions and also similar to object management in HLA [22]:

- Different publishers advertise their sets of titles available for subscription.
- Subscribers may subscribe to titles of their choice when they would like to.
- Publishers will then publish issues at regular intervals, which all subscribers will receive.
- All subscribers receive identical copies of an issue of a title.
- Publishers may also add titles at any given time to their collections. Cancellation of titles is not supported at present.
- A subscriber can read a copy of an issue as many times as they would like until a new one arrives.
- A subscriber may also elect to ignore old issues and only read the latest. Note that this is one of two supported modes, the second is an extension to the analogy (See next list).
- A publisher cannot change the content of an issue once it has been published and received by its subscribers.

The implementation of the publish-subscribe mechanism is somewhat extended over the analogy to allow more flexibility:

- Subscribers can select at what interval (rate) they want to receive issues. The maximum update rate is limited by the smallest time increment (frame) of the simulation.
- Subscribers can select if they want all issues of a given publisher, or just selected titles, without subscribing separately to each title.
- The subscriber may select what will happen with copies of issues that are received but not read. If kept, all copies from the oldest to the most recent have to be read to get to the latest issue. If not kept, the most recent copy is always available to be read. An in between mode is not supported.
- A specialised extension to support modelled communications between objects (modelled equipment or operators, not IPC related communications) is provided with additional parameters to support transmission functionality (status, delays, sender/receiver identification, etc.). Messages destined for transmission are passed immediately to the receiver where they are delayed in a cache to model the correct transmission delay, until delivery. The minimum transmission delay of a message is limited by the minimum time increment of the simulation. Messages are also rather delayed at the receiver than the transmitter, as the receiver knows, implicitly, its own position and, from the issue meta-data, the sender's position which are both required by the communications model.

Functionality as listed in the above two lists, are directly supported in the simulation architecture as part of the base object model and the simulation backbone, which is a set of

classes and functions providing the necessary mechanisms.

B. Peer-to-peer Processing Node Architecture

A peer-to-peer processing node architecture was ultimately selected above the client-server architecture, as the server may form a bottleneck due to the double latency and bandwidth usage for messages transmitted from a client to the server and then to the receiving client from the server (Figure 3(a)).

To minimise traffic at a server, an intermediate layer of servers were considered before using the peer-to-peer architecture. The intermediate servers (Figure 3(b)) have less traffic to route, and will only transmit messages to other intermediate servers via the top-level server if a receiving client requires it. The scheme is efficient, but has one major drawback: The architecture is not domain independent, as the clustering of clients per intermediate server requires prior knowledge of clients that can be grouped by type or anticipated traffic.

Note that the peer-to-peer architecture will result in a single latency for messages passed between nodes (indicated as peers in Figure 3(c)), but IPC connections have to be brokered or configured in some way before a simulation execution starts. In the client-server case, all clients connect to the same server. The peer-to-peer architecture suffers from the same domain knowledge challenge as the intermediate server solution, i.e. which models may be grouped for acceptable execution performance. The ultimate architecture would allow for both the automatic distribution of models across processing nodes, as well as automatically introducing intermediate server layers for optimal execution performance. Each processing node executes a subset of all objects (models, consoles and services). In the case where a single processing node is used, all objects are executed on it. As conservative time management is used in a time-stepped fashion, the slowest or most processing intensive object governs the global execution performance of the simulation. Load balancing is therefore necessary and is supported either as a static configuration or with dynamic load management [23]. The latter requires passing of objects in-process between processing nodes during run-time.

The peer-to-peer architecture is fully connected, thus each node is connected to each other node at start-up using TCP. This results in $\frac{n(n-1)}{2}$ connections, where n is the number of nodes (peers). For the client-server case the number of connections equals the number of clients. Connections between objects are made using a proprietary, binary-packed protocol, irrespective if objects are on the same processing node or not, or if only one processing node is used.

C. TCP Implementation Details

Specific TCP implementation tweaks to ensure lower message latency between nodes are discussed in this subsection.

TCP messages are grouped per destination node and sent off together instead of sending each message separately. Message latency still turned out to be a problem due to TCP's Nagel algorithm [24]. The Nagel algorithm usually improves bandwidth (saves on message header overhead) by caching short messages for a certain time-out or until they are big enough to fill a complete data packet before sending the data. Typically message groups were much smaller than the normal packet size of 1.5 kilobyte. Turning off the Nagel algorithm gave the simulation architecture complete control over message sending times which decreased latencies considerably. The communication model would cause a node to send information to every other node once every simulation time increment which means that the shorter the latencies the faster the simulation can execute.

The TCP sending buffer was also made bigger than the default to allow the simulation architecture to push messages into the sending buffer without blocking to allow the simulation to continue processing while the TCP operating system thread continues sending. This approach saves the overhead of implementing the simulation's TCP sending code in a separate processing thread.

To start a distributed simulation, all the nodes except the first node may be started up in any order. As soon as the first node is started it makes connections to all the other nodes, which in turn make connections to each other and finally start the simulation.

IV. Results

Initial experiments with a non-distributed and intermediate server-client (see Section III) architecture showed that in order to execute large enough simulations, distributed processing would be required to maintain soft real-time compatibility [25]. Approximately 6-8 processing nodes were estimated for real-time compliance, but less could be used, as the model loading metrics were very conservative. Between 40 and 100 objects, with varying levels of fidelity were anticipated. With the actual architecture, a fully populated scenario translates to 177 entities that require processing. The architecture is still efficient enough to execute this at approximately soft real-time. Of the entities, 160 are models, 8 consoles and 9 services.

Soft real-time execution is maintained by synchronising with the local processing node clock. Processing time is yielded not to exceed real-time. However, this only works when the processing nodes are not overloaded, i.e. able to process all models within a simulation time frame, otherwise extra processing nodes may be added. If this still does not help, soft real-time compatibility cannot be maintained.

Distributed performance tests were done with a processing intensive test object that takes exactly 1ms PC time to increment and publishes a single title which is a text string of length 512 bytes. Each test object subscribes to the titles of all the other test objects, including its own title. The communication setup is thus fully connected over all objects. A 100Hz closed loop distributed simulation is run over one to six machines in as fast as possible mode. Simulation distribution and communication overhead results are presented in Table I. The simulation frames are 10ms in length, equating to a 100Hz update rate, giving ten 1ms slots for a maximum of ten models per node to sustain soft real-time execution. It can be seen that a single node is very efficient, running at

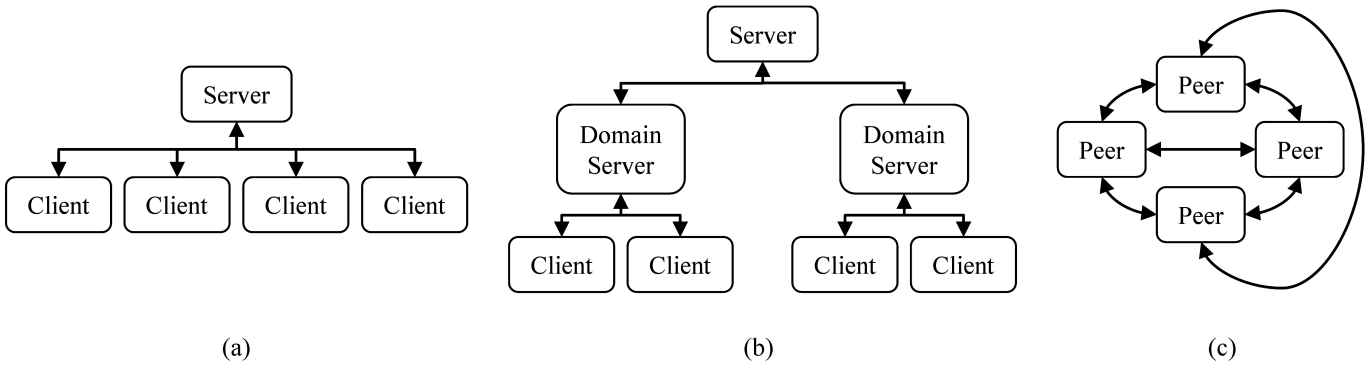


Fig. 3: Client-Server (a) Intermediate-Server (b) and Peer-to-Peer (c) Processing Node Architectures

99% of real time with 10 models which translates to a 1% communication overhead. Table I also shows that:

1) to run up to 9 test objects real time at least 1 node is required;
2) to run up to 18 test objects real time at least 2 nodes are required;
3) to run up to 24 test objects real time at least 3 nodes are required;
4) to run up to 32 test objects real time at least 4 nodes are required;
5) to run up to 35 test objects real time at least 5 nodes are required;
6) to run up to 42 test objects real time at least 6 nodes are required.

To run 42 test objects at real time at least 6 nodes are required running 7 objects each. This translates to an average communication overhead of just under 30%. Network usage was measured by using Windows XP's task manager network performance window. Note that the tests performed are worst case scenarios. All objects subscribe to all other objects and themselves (n^2 subscriptions) and the issue size is 512 bytes which is big enough for 21 double precision 3D coordinate triplets or a list of approximately 64 English words.

The simulation architecture has been successfully applied in an air defence simulation as a decision support tool and for standard operating procedure concept evaluation. It has been applied in extensions of the air defence simulation that include satellite-based optical and radar sensors for maritime surveillance concept development. It was also used for integration with various external systems, including situational air picture systems, operator console mock-ups, air traffic radars, flight simulators and similar simulations.

V. Future Work

A key challenge with soft real-time simulations is what should be done if the simulation slips on world-time? This often occurs, as models tend to have spurious high processing requirements. What techniques should be used to catch up again on world-time, specifically when a simulation is connected to external systems, such as a flight simulator or air picture systems? One solution is to use innovative schemes in console implementations to external systems to cater for data that arrives at the wrong-time, i.e. too late, or to too early. In the first case, prediction algorithms are necessary and in the latter buffering schemes.

Future work includes conducting comparative studies between peer-to-peer, intermediate server and client-server architectures. There is also ample opportunity for load balancing research: How to measure model loading per processing node efficiently and effectively, and load balancing algorithms.

VI. Conclusion

The ability to execute an entire simulation in an all-in-one mode on a single processing node (desktop computer) is a key advantage in how the simulation architecture is used. It is efficient and quick to configure scenarios for simulation, and to visually verify them using peripheral two and three dimensional viewers. Similar techniques are used to verify and validate newly integrated models, consoles or services. It is then merely a matter of changing a configuration to execute the simulation distributed to achieve soft real-time execution. The simulation architecture is suitable for parallel execution on a small to medium scale infra-structure.

The publish-subscribe architecture is very flexible in terms of objects and connection management. However, care should be taken to adhere to a standard way of implementing objects and not to abuse the flexibility.

The soft real-time performance figures obtained with worst-case object loadings indicate that the architecture is adequate for a small number of nodes with less than 10 objects per node. It is expected that network overheads will limit scalability to less than 100 objects in total, and therefore, the architecture is not considered to be scalable for large simulations (100's of objects), requiring soft real-time performance.

Author Biographies

BERNARDT DUVENHAGE has been with the CSIR within the Mathematical and Computational Modelling Research Group, that's part of the Defence, Peace, Safety and Security (DPSS) Research Group, since January 2004. Past responsibilities have included the development of a distributed simulation architecture, development of an optimized Line of

TABLE I: Distribution and Communication Overhead Results

Number of Nodes	Objects per Node	Issue Size	Percentage Real-Time	Network Utilization	Total Objects
1	7	512 bytes	142%	0%	7
2	7	512 bytes	132%	6%	14
3	7	512 bytes	126%	12%	21
4	7	512 bytes	117%	16%	28
5	7	512 bytes	110%	20%	35
6	7	512 bytes	102%	25%	42
1	8	512 bytes	124%	0%	8
2	8	512 bytes	116%	7%	16
3	8	512 bytes	110%	14%	24
4	8	512 bytes	100%	18%	32
5	8	512 bytes	92%	23%	40
6	8	512 bytes	85%	27%	48
1	9	512 bytes	110%	0%	9
2	9	512 bytes	104%	8%	18
3	9	512 bytes	99%	15%	27
4	9	512 bytes	90%	21%	36
5	9	512 bytes	81%	26%	45
6	9	512 bytes	75%	27%	54
1	10	512 bytes	99%	0%	10
2	10	512 bytes	93%	9%	20
3	10	512 bytes	88%	17%	30
4	10	512 bytes	77%	23%	40
5	10	512 bytes	70%	28%	50
6	10	512 bytes	60%	32%	60

Sight (LOS) algorithm and LOS service, development of a terrain attitude and altitude service, development of models in cooperation with OEMs and development of a 3D analysis tool based on the Open source Scene Graph (OSG). He completed his BSc(hons) in Computer Science in 2005 and is currently pursuing an MSc in Computer Science on real time simulation architectures.

HERMAN LE ROUX has been with the South African Council for Scientific and Industrial Research since April 1998 and is at present a Principal Engineer in the Mathematical and Computational Modelling Research Group. He is involved in Modelling and Simulation-based Acquisition Decision Support, specifically for the South African National Defence Force. Interests include information fusion, biometrics, artificial intelligence and software engineering. Le Roux completed a Masters Degree in Computer Engineering at the University of Pretoria in 1999 and is currently pursuing a PhD in Information Fusion.

ACKNOWLEDGEMENTS

The authors would like to thank Dr Jackie Phahlamohlaka, Cobus Nel, Anita Louis and Arno Duvenhage, all colleagues at the CSIR, for their valuable inputs, as well as the Armaments Corporation of South Africa for supporting this work.

REFERENCES

[1] DOD, "Department of Defense: Modelling and Simulation Master Plan," Under Secratary of Defense for Acquisition and Technology, DoD 5000.59-P, Alexandria, VA, October 1995.

[2] J. J. Nel and H. J. Baird, "The evolution of M&S as part of smart acquisition using the SANDF GBADS programme as example," in *Twelfth European Air Defence Symposium*, Shrivenham, June 2005.

[3] W. H. le Roux, "Implementing a low cost distributed architecture for real-time behavioural modelling and simulation," in *Proceedings of the 2006 European Simulation Interoperability Workshop*. Stockholm: Simulation Interoperability Standards Organization, June 2006.

[4] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, and I. Wakeman, "Towards yet another peer-to-peer simulator," FOURTH INTERNATIONAL WORKING CONFERENCE PERFORMANCE MODELLING AND EVALUATION OF HETEROGENEOUS NETWORKS, West Yorkshire, September 2006.

[5] A. Montresor, G. D. Caro, and P. E. Heegaard, "Architecture of the simulation environment," Information Society Technologies, Italy, Project Report IST-2001-38923, January 2004.

[6] S. Cheon, C. Seo, S. Park, and B. P. Zeigler, "Design and implementation of distributed DEVS simulation ina peer to peer network system," in *Proceedings of the 2004 Military, Government and Aerospace Simulation Symposium*, Virginia, April 2004.

[7] B. Gedik and L. Liu, "A scalable peer-to-peer architecture for distributed information monitoring applications," *IEEE Transactions on Computers*, vol. 54, no. 6, pp. 767–783, June 2006.

[8] G. D. Costa, "Peer-to-peer simulation," Presentation at the Federal University of the Rio Grande Do Sul, Porto Alegre, 2002.

[9] Y. Kawahara, H. Morikawa, and T. Aoyama, "A peer-to-peer message exchange scheme for large scale networked virtual environments," Proceedings of the 8th IEEE International Conference on Communications Systems (ICCS 2002), Amsterdam, April 2002.

[10] S.-P. A. van Houten and P. H. M. Jacobs, "An architecture for distributed simulation games," in *Proceedings of the 2004 Winter Simulation Conference*, R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, Eds., Washington DC, December 2004.

[11] S. Giesecke, T. Warns, and W. Hasselbring, "Availability simulation of peer-to-peer architectural styles," in *Proceedings of the 2005 workshop on Architecting dependable systems (WADS 2005)*. Waterloo: ACM Press, 2005, pp. 1–6.

[12] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori, "P2prealm peer-to-peer network simulator," in *Proceedings of the 11th Intenational Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, June 2006, pp. 93–99.
[13] R. Weatherly, D. Seidel, and J. Weissman, "Aggregate Level Simulation Protocol," Presented at the 1991 Summer Computer Simulation Conference, Baltimore, July 1991.
[14] "IEEE standard for information technology - protocols for distributed interactive simulations applications," IEEE Std 1278-1993, 1993.
[15] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, ser. Artificial Intelligence. Upper Saddle River: Prentice Hall, 1999.
[16] J. S. Steinman and D. R. Hardy, "Evolution of the standard simulation architecture," http://www.modelingandsimulation.org/issue10/SISO/steinman.html, vol. 3, nr. 2 , issue 10, April-June 2004, Accessed 16 January 2007.
[17] O. Dalle, "OSA: an open Component-based architecture for Discrete-event Simulation," INRIA, Tech. Rep. RR-5762, February 2006, available from http://www.inria.fr/rrrt/rr-5762.html.
[18] P. A. Hawley and T. J. Urban, "An object-oriented simulation architecture," AIAA Modeling and Simulation Technologies Conference and Exhibit, Providence, August 2004.
[19] D. Brutzman, M. Zyda, J. M. Pullen, and K. L. Morse, "Extensible modeling and simulation framework (XMSF) challenges for web-based modeling and simulation," TECHNICAL CHALLENGES WORKSHOP, STRATEGIC OPPORTUNITIES SYMPOSIUM, www.movesinstitute.org/xmsf, San Diego, October 2002.
[20] J. H. S. Roodt, L. Berglund, and P. Klum, Worksession between FOI and CSIR, Linköping, 2001.
[21] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: A network software architecture for large scale virtual environments," *Presence*, vol. 3, no. 4, 1994.
[22] "High Level Architecture interface specification: Version 1.3," U.S. Department of Defense, April 1998.
[23] B. Duvenhage, "The contribution of static and dynamic load balancing in a real-time distributed air defence simulation," 2007, to be submitted to the Summer Computer Simulation Conference, San Diego.
[24] B. A. Forouzan, *TCP/IP Protocol Suite*, 2nd ed., ser. Forouzan Networking. Boston: McGraw-Hill, 2003.
[25] B. Duvenhage and W. H. le Roux, "MobADS: TCP-based distributed simulation architecture investigation," Council for Scientific and Industrial Research, Tech. Rep. DEFT-MSADS-00151, July 2004.

RUBLX : A RUBY-BASED BATCH LANGUAGE FOR XGRID

Tetsuya SUZUKI
Kiyoto HAMANO
Department of Electronic Information Systems
Shibaura Institute of Technology
Minuma, Saitama city, Saitama,
337-8570, Japan
Email: {tetsuya, m107076}@sic.shibaura-it.ac.jp

Abstract— **We present a Ruby-based batch language for Xgrid and its processor. Xgrid is an environment for distributed and parallel computing on the Mac OS X operating system, and Ruby is an object-oriented programming language for general purposes. In the standard Xgrid environment, jobs in batch files are statically defined by an XML-based language, and submitted jobs are managed by their ID numbers. It is not easy for human to read and write XML-based batch files and to manage jobs by ID numbers. In our approach, jobs in batch files can be dynamically defined by a Ruby-based language, and submitted jobs can be managed by their logical names. Semantic checks and consistency managements are also done at submission in our approach. Our approach syntactically and semantically makes it easy to use Xgrid.**

Keywords— **Grid and Cluster Computing, Languages**

I. Introduction

Xgrid[1] is an environment for distributed and parallel computing on the Mac OS X operating system. We use Xgrid for Web mining and metadata generation[5], which need much computation. By the standard method, jobs for Xgrid can be submitted by an XML-based batch language and are managed by their job ID numbers.

We have difficulties in reading and writing the XML-based batch files and managing jobs. XML tags in batch files and job management by IDs are obstacles for them.

To solve the problems, we present a Ruby-based batch language for Xgrid and its processor. Ruby[2] is an object-oriented programming language. By our method, jobs can be submitted by concise batch files and be managed by symbolic names.

The organization of this paper is as follows. In section 2 we explain Xgrid and its batch language. We present our approach using examples in section 3, and compare it with other approaches in section 4. In section 5 we state our conclusion. We explain the detail of our batch language in appendix.

II. Grid and the batch language

A. Xgrid

Xgrid mainly consists of three kinds of software: clients, a controller and agents. A *client* is a program to submit jobs to a controller and receive the results from it. A *job* is a set of tasks, and a *task* is a program executed in Xgrid. A *controller* is a program to receive jobs from clients, split them into tasks and send them to agents. An *agent* is a program to execute assigned tasks and return the results to the controller. The results are sent to clients via the controller. They can work on different computers connected by local area network.

```
% xgrid -h xgridcontroller -p pass
        -job batch bc.plist
```

Fig. 1. Job submission by 'xgrid'

```
% xgrid -h xgridcontroller -p pass
        -job results -id 381
```

Fig. 2. Retrieval of the results by 'xgrid'

The standard client program 'xgrid', which is invoked from command line interface, provides two methods to submit jobs. In the first method we specify a job, which consists of a program and its command line arguments, in the command line arguments for 'xgrid'. We can submit a single task job only at once by the method. In the second method we specify a batch file in the command line arguments for 'xgrid'. A *batch file* is a file to specify jobs and their tasks. In batch files we can also describe jobs which must finish before a job starts, and tasks which must finish before a task starts. We call such relations *dependency relationships*. We can submit multi-task jobs at once by the method. The client program 'xgrid' also provides methods to manage jobs and retrieve their results using job ID numbers. For example, we can stop, delete and restart jobs.

Fig.1 and Fig.2 show how to use the 'xgrid' command from command line. Fig.1 is an example of job submission where the argument 'bc.plist' is a batch file name. Fig.2 is an example of retrieval of the results where the argument '381' is the job ID whose results is retrieved. In both cases the '-h' and the '-p' options specify a controller's host name and a password to connect it respectively.

B. The standard batch language

The standard batch language for Xgrid is based on XML with key/value structure. We explain the language using Fig.3 in the following.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

Fig.3 shows a batch file in the batch language. It defines a job with a task which executes a command line '/usr/bin/bc -q bc_exp.txt' in Xgrid where a calculator program '/usr/bin/bc' reads an expression '1+2' from the script file 'bc_exp.txt' of Fig.4 and outputs the value of the expression to its standard output. The script file 'bc_exp.txt' is defined from the line 8 to the line 18 of Fig.3. The contents of the script file is embedded in the line 13 of Fig.3 as a base64-encoded string. Base64[3] is an encoding method which translates a byte stream to a US-ASCII string. The task is defined from the line 19 to the line 31 of Fig.3. The command path and its command line arguments are defined there.

As shown in Fig.3, users are responsible for consistency managements such that a job definition must include all file definitions which tasks in the job will refer to.

The batch language provides task prototype for concise task definitions though it is not used in Fig.3.

C. Problems in use of Xgrid

The followings are problems in use of Xgrid.

1. XML tags are obstacles for human to read and write batch files.
2. Files referred by tasks must be embedded as base64-encoded strings in batch files.
3. Jobs can not be dynamically determined by batch files at submission. In cases such that a job to execute many of a program with different parameters is described, the batch file can be more concise by dynamically determined jobs.
4. Jobs are managed by their ID numbers.
5. Dependency relationships among jobs in a same batch file can not be specified. The relationships must be specified by job IDs determined at submission which are never known at describing the batch file.
6. Xgrid users are responsible for semantic consistency check of batch files.

III. A Ruby-based batch language and its processor

To solve the problems pointed out in the previous section, we propose a Ruby-based batch language for Xgrid (RuBLX) and a client program 'rxgrid' for RuBLX. In this section, we explain the design principles and examples of batch files in RuBLX. We explain the detail of our batch language in appendix.

A. Design principles

The followings are design principles for our approach.

1. XML tags are not used in batch files.
2. Base64 encoding are not needed in batch files.
3. Jobs can be dynamically determined.
4. Jobs can be managed by symbolic names.
5. Dependency relationships among jobs in a same batch file can be specified.

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <!DOCTYPE plist PUBLIC
"-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
 3: <plist version="1.0">
 4:  <array>
 5:   <dict>
 6:    <key>name</key>
 7:    <string>job1</string>
 8:    <key>inputFiles</key>
 9:    <dict>
10:     <key>bc_exp.txt</key>
11:     <dict>
12:      <key>fileData</key>
13:      <data>MSArIDIKcXVpdAo=
14:      </data>
15:      <key>isExecutable</key>
16:      <string>NO</string>
17:     </dict>
18:    </dict>
19:    <key>taskSpecifications</key>
20:    <dict>
21:      <key>bc</key>
22:      <dict>
23:       <key>command</key>
24:       <string>/usr/bin/bc</string>
25:       <key>arguments</key>
26:       <array>
27:        <string>-q</string>
28:        <string>bc_exp.txt</string>
29:       </array>
30:      </dict>
31:    </dict>
32:   </dict>
33:  </array>
34: </plist>
```

Fig. 3. An XML-based batch file 'bc.plist'

```
1 + 2
quit
```

Fig. 4. A script file 'bc_exp.txt'

6. Semantic checks and consistency management are automatically done.

We use the Ruby programming language as the basis of our batch language because Ruby can evaluate a string as its program. Our client program 'rxgrid' implemented in Ruby evaluates an RuBLX batch file as a Ruby program for lexical analysis, syntax analysis and semantic analysis. It then generates XML-based batch files where base64 encoding and semantic check are automatically done, and submits them using 'xgrid'. The client generates an XML-based batch file for each job in a same RuBLX batch file to enable to specify dependency relationships among jobs in the RuBLX batch file. It also generates a map file, which records the correspondence between job IDs and symbolic job names, at submission. The client provides job management methods by symbolic job names using map files.

B. Examples

We explain an overview of our approach and show that our approach solves the problems pointed out in the previous section using three examples.

```
 1: file "bc_exp.txt" do |t|
 2:   t.agentPathName = "bc_exp.txt"
 3:   t.localPathName = "bc_exp.txt"
 4:   t.isExecutable = false
 5: end
 6:
 7: task "bc" do |t|
 8:   t.command = "/usr/bin/bc"
 9:   t.arguments = ["-q", "bc_exp.txt"]
10:   t.refersTo = ["bc_exp.txt"]
11: end
12:
13: job "job1" do |t|
14:   t.tasks = ["bc"]
15: end
```

Fig. 5. An RuBLX batch file 'bc.rb'

```
381,job1
```

Fig. 6. A map file 'bc_map.csv'

```
% rxgrid -h xgridcontroller -p pass
         -job batch bc.rb
```

Fig. 7. Job submission

```
% rxgrid -h xgridcontroller -p pass
         -job results -id job1
         -map bc_map.csv
```

Fig. 8. Retrieval of the results

The first example is shown in Fig.5. It is an example of a batch file in RuBLX. The job described there is the same as the job of Fig.3. Files, tasks and jobs have logical names as identifiers, and are defined as follows.

- A file is defined from the line 1 to the line 5. It starts with a keyword 'file' followed by a logical file name 'bc_exp.txt' and a do-end block with a parameter 't'. The followings are defined in the block: a path name of the file on agent machines, the contents of the file by a local file and whether it is executable or not.
- A task is defined from the line 7 to the line 11. It starts with a keyword 'task' followed by a logical task name 'bc' and a do-end block with a parameter 't'. The followings are defined in the block: a path name of a command, command line arguments for it and a file referred by it,
- A job is defined from the line 13 to the line 15. It starts with a keyword 'job' followed by a logical job name 'job1' and a do-end block with a parameter 't'. A task in the job is defined in the block.

The batch file in Fig.5 is more concise than the batch file in Fig.3. There is no XML tag in Fig.5, and the number of lines in Fig.5 is 15 while that in Fig.3 is 34.

A consistent management is done at submission of the batch file. The job definition in the generated XML-based batch file includes a file definition which the task 'bc' refers to though the file definition is not referred in the job definition in the RuBLX batch file.

Fig.6 shows a map file generated at submission of the batch file of Fig.5. The map file indicates that the ID number of the job 'job1' is 381.

Fig.7 and Fig.8, which correspond to Fig.1 and Fig.2 respectively, show how to use our processor 'rxgrid'. Fig.7 shows an example of job submission where the command line argument 'bc.rb' is a batch file name. Fig.8 shows an example of retrieval of the results where the command line arguments 'job1' and 'bc_map.csv' are the job name whose results is retrieved and a map file respectively.

The second example is shown in Fig.9. It is an example of a batch file with dependency relationships among jobs. Three jobs 'job0', 'job1' and 'job2' are defined there.

- The job 'job0' is a previously submitted job with the job ID 333. The job ID can be specified by a pair of a logical job name and a map file as shown in appendix.
- The job 'job1' is a job with a task 'echo1'.
- The job 'job2' is a job with a task 'echo2' which starts after two jobs 'job0' and 'job1' are done. The dependency is defined by 't.dependsOnJobs' in the block.

Our client program takes account of both dependency relationships among jobs and those among tasks. It does topological sort on jobs for submission. If it finds either cyclic dependency relationships for jobs or those for tasks, it declares errors.

The third example is shown in Fig.10. It is an example of a batch file where the number of tasks in a job are dynamically determined at submission.

In the batch file, a task is defined for each file whose name ends with '.txt' in a current directory. The files are collected by a standard Ruby library 'Dir' in the line 1 of Fig.10. Each task calculates the value of an expression in the file using '/usr/bin/bc'. Variables, arrays, flow controls and a standard Ruby library are used in the batch file because it is not known how many files will exist in a current directory at submission when the batch file is written.

Definitions of files, tasks and jobs are basically declarative in RuBLX. The order of definitions is not significant. Procedural description, however, can be used as this example.

Templates can be used for definitions in RuBLX though some programming skill is needed. For example, a template for a task is used from the line 16 to the line 20 in Fig.10 where 'taskName' and 'f.to_s' are used as parameters for the template.

IV. Comparison

We compare our approach with PyXG[1] because our approach solves problems of the standard Xgrid

```
 1: task "echo1" do |t|
 2:   t.command = "/bin/echo"
 3:   t.arguments = ["1"]
 4: end
 5:
 6: task "echo2" do |t|
 7:   t.command = "/bin/echo"
 8:   t.arguments = ["2"]
 9: end
10:
11: job "job0" do |t|
12:   t.id = 333
13: end
14:
15: job "job1" do |t|
16:   t.tasks = ["echo1"]
17: end
18:
19: job "job2" do |t|
20:   t.tasks = ["echo2"]
21:   t.dependsOnJobs = ["job0", "job1"]
22: end
```

Fig. 9. An RuBLX batch file with dependency relationships among jobs

```
 1: filelist = Dir.glob("*.txt")
 2:
 3: filelist.each do |f|
 4:   file f.to_s do |t|
 5:     t.agentPathName = f.to_s
 6:     t.localPathName = f.to_s
 7:     t.isExecutable = false
 8:   end
10: end
11:
12: taskNames = []
13: filelist.each do |f|
14:   taskName = "bc" + f.to_s
15:   taskNames = taskNames | [taskName]
16:   task taskName do |t|
17:     t.command = "/usr/bin/bc"
18:     t.arguments = ["-q", f.to_s]
19:     t.refersTo = [f.to_s]
20:   end
21: end
22:
23: job "job1" do |t|
24:   t.tasks = taskNames
25: end
```

Fig. 10. An RuBLX batch file with dynamically defined tasks

environment which we pointed out as shown in the previous section, and PyXG also uses a programming language to specify and submit Xgrid jobs as our approach.

PyXG is a module for the Python programming language[4], which enables to submit jobs to Xgrid controllers and manage them from Python programs. Python is an object-oriented programming language. Fig.11 shows a program with PyXG. In the program, not only job construction steps (the lines 5 and 6) but also steps for connecting to a controller (the lines 2, 3 and 4) and a step for submission (the line 7) are described.

```
1: from xg import *
2: conn = Connection(
           hostname='xgridcontroller',
           password='pass')
3: cont = Controller(conn)
4: g = cont.grid(0)
5: js = JobSpecification()
6: js.addTask('/usr/bin/bc',
              args='bc_script.txt')
7: j = g.batch(js)
```

Fig. 11. A Python program with PyXG

The main differences between PyXG and our approach are as follows.

1. Programs with PyXG are completely procedural while our batch files are basically declarative. The order of definitions is not significant in our approach. Procedural descriptions are used in our batch files if needed as shown in Fig.10. In PyXG, semantic checks and the ordering of job submission must be procedurally described. In our approach, they are done automatically.
2. Programs with PyXG includes everything related to Xgrid while our batch files consist of job specifications only.

The interested reader is referred to [1] for other Xgrid client programs.

V. Conclusions

We proposed a Ruby-based batch language for the grid computing environment Xgrid, and a client program for the language. They solve the problems about Xgrid: XML tags as obstacles for human to read and write batch files, consistency management of batch files by users, job management by job ID numbers, and so on. Users with some programming skill can describe batch files using templates. Our approach makes it easy to use Xgrid.

Acknowledgements

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 18700035 from 2006 to 2007.

References

[1] Baden Hughes. Building computational grids with apple's xgrid middleware. In Rajkumar Buyya, Tianchi Ma, Reihaneh Safavi-Naini, Chris Steketee, and Willy Susilo, editors, *ACSW Frontiers*, volume 54 of *CRPIT*, pages 47–54. Australian Computer Society, 2006.

[2] Brian Marick. *Everyday Scripting with Ruby.* Pragmatic Bookshelf, 2007.

[3] David Wood. *Programming Internet Email*. Oreilly & Associates Inc, 1999.
[4] Mark Lutz and David Ascher. *Learning Python*. Oreilly & Associates Inc, 2003.
[5] Tetsuya Suzuki and Takehiro Tokuda. A system for landscape photograph localization. In *ISDA (1)*, pages 1080–1085. IEEE Computer Society, 2006.

Appendix

We explain the detail of our batch language. A batch file includes one or more job definitions, one or more task definitions and zero or more file definitions. The order of definitions is not significant. It can also include any Ruby code anywhere.

Fig.12, Fig.13, Fig.14 and Fig.15 show the syntax of a file definition, a task definition and job definitions respectively. In the figures, non terminal symbols are enclosed with '<' and '>'. A pair of parenthesises followed by a question mark '(X)?' means X is optional. A pair of parenthesises with a vertical bar between them '$(X \mid Y)$' means X or Y.

Each definition has a do-end block with a parameter <*PARAM*>, and the detail of each definition is given there. The order of description in such a do-end block is not significant. A same identifier must be used for <*PARAM*> in a same do-end block.

In the following, we explain the syntax of a file definition, a task definition and a job definition in this order.

Fig.12 shows the syntax of a file definition. A file definition starts with a keyword 'file'. It takes two arguments: a string for a logical file name(<*LOGICAL_FILE_NAME*>) and a block with a parameter. The block specifies the detail of the file. In the following, we use 't' as the parameter. In the block, the value of 't.agentPathName' specifies a path of the file on an agent machine. The content of the file is specified by 't.localPathName' or 't.contents'. The value of 't.localPathName' <*PATH_ON_LOCAL*> specifies the contents by the path of a local file. The value of 't.contents' <*STRING*> specifies the contents by a string. Both of them can not be specified at once. The value of 't.isExecutable' <*EXECUTABLE*> specifies whether the file is executable or not: 'true' for executable, 'false' for not-executable.

Fig.13 shows the syntax of a task definition. A task definition starts with a keyword 'task'. It takes two arguments: a string for a logical task name(<*LOGICAL_TASK_NAME*>) and a block with a parameter. The block specifies the detail of the task. In the following, we use 't' as the parameter. In the block, the value of 't.command' <*PATH_OF_COMMAND*> specifies a path of a command which will run on an agent machine. The value of 't.arguments' <*COMMAND_ARGUMENT_LIST*> specifies command line arguments by an array. The value of 't.environment' <*ENVIRONMENT_HASH*> specifies environment variables and their values by a hash. The keys of the hash are names of environment variables, and their values are values of the environment variables. The value of 't.inputStream' <*LOGICAL_FILE_NAME*> specifies a logical file name whose contents are used as the standard input. The value of 't.dependsOn' <*LOGICAL_TASK_NAME_LIST*> specifies logical task names which the task depends on by an array. The value of 't.refersTo' <*LOGICAL_FILE_NAME_LIST*> specifies logical file names by an array, each of which the task will read. The value of 't.inputFileMap' <*INPUT_FILE_MAP_HASH*> specifies the correspondence between file paths on agents and the contents for this task only by a hash. The keys of the hash are file paths and their values are logical file names.

Fig.14 and Fig.15, which are for previously submitted jobs and jobs to be submitted respectively, show the syntax of a job definition.

In both cases, a job definition starts with a keyword 'job'. It takes two arguments: a string for a logical job name(<*LOGICAL_JOB_NAME*>) and a block with a parameter. The block specifies the detail of a job. In the following, we use 't' as the parameter.

The detail of previously submitted jobs are defined in the block of Fig.14 as follows. The value of 't.id' specifies a previously submitted job ID. The previously submitted job ID is given either by an integer or by a pair of a logical job name and a map file. The pair is specified by 'jobId(<*LOGICAL_JOB_NAME*>, <*MAP_FILE_PATH*>)' where <*LOGICAL_JOB_NAME*> is a logical job name and <*MAP_FILE_PATH*> is the path of a map file which includes the logical job name.

The detail of jobs to be submitted are defined in the block of Fig.15 as follows. The value of 't.mail' <*MAIL_ADDRESS*> specifies an e-mail address to which an e-mail is sent when the job status is changed. The value of 't.taskMustStartSimultaneously' <*TASK_MUST_START_SIMULTANEOUSLY*> specifies whether tasks must start simultaneously or not by a boolean value: 'true' is for yes, and 'false' is for no. The value of 't.minimumTaskCount' <*MINIMUM_TASK_COUNT*> specifies the minimum number of tasks which are needed to start at the same time. The value of 't.dependsOnJobs' <*LOGICAL_JOB_NAME_LIST*> specifies logical job names which the job depends on by an array. The value of 't.files' <*LOGICAL_FILE_NAME_LIST*> specifies logical file names used in the job by an array. The value of 't.tasks' <*LOGICAL_TASK_NAME_LIST*> specifies logical task names in the job by an array.

```
file <LOGICAL_FILE_NAME> do | <PARAM> |
     <PARAM> .agentPathName = <PATH_ON_AGENT>
     ( <PARAM> .localPathName = <PATH_ON_LOCAL> | <PARAM> .contents = <STRING> )
     ( <PARAM> .isExecutable = <EXECUTABLE> ) ?
end
```

Fig. 12. The syntax of a file definition

```
task <LOGICAL_TASK_NAME> do | <PARAM> |
     <PARAM> .command = <PATH_OF_COMMAND>
     ( <PARAM> .arguments = <COMMAND_ARGUMENT_LIST> ) ?
     ( <PARAM> .environment = <ENVIRONMENT_HASH> ) ?
     ( <PARAM> .inputStream = <LOGICAL_FILE_NAME> ) ?
     ( <PARAM> .dependsOn = <LOGICAL_TASK_NAME_LIST> ) ?
     ( <PARAM> .refersTo = <LOGICAL_FILE_NAME_LIST> ) ?
     ( <PARAM> .inputFileMap = <INPUT_FILE_MAP_HASH> ) ?
end
```

Fig. 13. The syntax of a task definition

```
job <LOGICAL_JOB_NAME> do | <PARAM> |
    <PARAM> .id = ( <PREVIOUSLY_SUBMITTED_JOB_ID> |
                              jobId( <LOGICAL_JOB_NAME> , <MAP_FILE_PATH> ) )
end
```

Fig. 14. The syntax of a job definition (case 1)

```
job <LOGICAL_JOB_NAME> do | <PARAM> |
    ( <PARAM> .mail = <MAIL_ADDRESS> ) ?
    ( <PARAM> .taskMustStartSimultaneously = <TASK_MUST_START_SIMULTANEOUSLY> ) ?
    ( <PARAM> .minimumTaskCount = <MINIMUM_TASK_COUNT> ) ?
    ( <PARAM> .dependsOnJobs = <LOGICAL_JOB_NAME_LIST> ) ?
    ( <PARAM> .files = <LOGICAL_FILE_NAME_LIST> ) ?
    <PARAM> .tasks = <LOGICAL_TASK_NAME_LIST>
end
```

Fig. 15. The syntax of a job definition (case 2)

EXPERIENCES WITH ASPECT-BASED PARALLELIZATION OF SCIENTIFIC CODE

Manuel Díaz, Sergio Romero, Bartolomé Rubio, Enrique Soler and José M. Troya
Department of Languages and Computer Science
University of Málaga
29071, Málaga, SPAIN
E-mail: mdr@lcc.uma.es

KEYWORDS
Languages, Object-Oriented Programming & Design.

ABSTRACT

This paper discusses the use of Aspect-Oriented Programming (AOP) to support the parallelization of scientific code. The idea is to develop parallelism concerns in separate aspects so that the weaving process can inject the code structures which allow the sequential scientific core to be executed in parallel. A series of advantages can initially be derived from this aspect-based approach. As parallelism is modularized into separate units, the code-tangling level is reduced. The parallel version is developed by reusing the code pieces which implement the numerical computation sequentially. Moreover, different aspects can be written to adapt the application to different high-performance environments. This paper describes some experiences with the code parallelization using aspects. Specifically, the integration of both task and data parallelism in the context of two parallel scenarios (i.e. multithreading and message-passing) is addressed in a case study. The work is an attempt to assess the benefits and limitations of applying AOP for these purposes.

INTRODUCTION

Scientific software frequently exploits parallelism for achieving high-performance when tackling large-scale and realistic engineering problems. Implementations are typically based on parallel programming libraries (e.g. PThreads, MPI, PVM) which provide the developer with a set of primitives to code parallelism concerns. However, code-tangling problems often arise in these applications as the statements describing the numerical computation are mixed with those expressing parallelism.

Aspect-Oriented Programming (Kiczales et al. 1997) helps the developer to achieve the separation of concerns, especially in those situations when such concerns cut across multiple parts (clases, components, modules) of a system. The modular management of cross-cutting concerns leads to a simpler application code which is easier to develop and maintain and has a greater potential for reuse. A well-modularized cross-cutting concern is called an aspect.

The management of high-performance concerns using AOP is emerging as a promising line of research. In (Harbulot and Gurd 2004), the separation of the parallel structure cross-cutting Java scientific applications is addressed using the general-purpose language AspectJ (Kiczales et al. 2001). The work in (Harbulot and Gurd 2006) describes an extension of the AspectJ join point model aimed at allowing loops to be parallelized without re-factoring the base-code. A methodology for the modular development of parallel programs which is based on the composition of multiple fine-grain aspects such as concurrency, partition, distribution and so on, is discussed in (Sobral 2006). The proposal in (Díaz et al. 2005) is focused on a component framework for the efficient development of high-performance applications. Specific concerns which affect a set of scientific components, such as the communication scheme underlying the application, are modeled into a special type of entities called aspect components.

This paper focuses on the idea that scientific programs can (possibly) be written sequentially in a way that enables parallelism to be added later, for instance, once the code has been tested and debugged. Although the parallelization of a scientific program may require many changes of diverse nature in different parts of its code, we can consider the effects of these changes as the result of weaving what we have called parallelization aspects into the sequential scientific core (i.e. the part in charge of the numerical computation). The bodies of these aspects will implement additional functionalities such as distribution, communication and synchronization, in order to enable the sequential core to be executed in parallel.

We can anticipate a series of advantages derived from this aspect-based parallelization approach:

- Parallel statements don't obscure the mathematical model, as the former are isolated into aspects, which results in a reduction of code-tangling.
- The parallel application is set-up by weaving aspects into the sequential core, promoting core reuse.
- The applied parallelism model can be replaced simply selecting different aspects to be woven.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

Despite the potential benefits, this approach may also lead to some serious shortcomings. The characteristics of the aspect-oriented language used determine the type of interaction between aspects and base-code. For instance, in a general-purpose language such as AspectJ, the interactions are mainly based on intercepting method calls (pointcuts plus advice code), augmenting data structures (introductions), and so on. However, the parallelization of code following these mechanisms can be very difficult unless the programs are written assuming (explicitly) the fact that the code pieces which will be parallelized must be accessed and managed through valid join points. This leads us to consider new sequential designs which include sets of interfaces and/or classes aimed at allowing aspects to inject parallelism. In some sense, the parallelization concern must be somehow considered from the initial design of the code. Obviously, this represents an extra effort in the development of the sequential scientific core.

This work is an attempt to assess the feasibility and suitability of the approach. The purpose is to describe a real experience which allows us to determine the overall advantages and limitations of using aspects for the code parallelization. At first, we describe a sequential program which computes the 2D-Fast Fourier Transform (Briham 1988) of a collection of matrices. Then, aspect weaving is used to integrate both task and data parallelism into the scientific code. This is carried out in the context of two different parallel scenarios: multithreading with shared memory, and message-passing based on MPI.

The implementation is based on the aspect-oriented language AspectC++ (Spinczyk et al. 2002). In AspectC++, almost all the language elements are efficiently implemented at compile-time, which makes the tool more suitable for the development of high-performance concerns than other approaches using Java-based dynamic weaving such as AspectJ.

The paper is structured as follows. Section 2 provides an overview of AspectC++. The case study is presented in section 3, where the sequential solution is described. Sections 4 and 5 provide an in-depth description of the parallelization using multithread programming and MPI, respectively. The paper finishes with some conclusions.

ASPECTC++ FUNDAMENTALS

AspectC++ is a general purpose language extension to C++ for the support of Aspect-Oriented Programming. An aspect can be understood as a modularized unit that implements a cross-cutting concern. The points at which an aspect can interfere with the base-code are called join points.

A pointcut identifies a set of join points. Pointcuts are described by means of pointcut expressions, which can refer to combinations of static program entities, such as classes, functions or namespaces, and other points in the control flow of the program. For instance, the expression:

```
execution("void Dialog::set%(...)")
```

refers to the execution of any method of `Dialog` having both a name beginning with `set` and `void` as return type. In the match expression, `%` is used as a wildcard symbol and `...` represents any sequence of arguments. A pointcut declaration allows a pointcut to be named so that it can be reused in different parts of the program.

The advice is the mechanism which defines the way the aspect affects the base-code. An advice declaration indicates the block of code to be executed when specific join points are reached. The advice code can be executed before, after, or both before and after (i.e. around) the join point. For instance, the following advice can be used to trace the execution of "setter" methods in `Dialog`:

```
advice execution("void Dialog::set%(...)") && that(d)
  : before(Dialog &d) {
  cout << "Dialog:" << d.name
       << " will be modified." << endl;
}
```

The code in the example above is triggered just before the method execution. The function `that()` binds a variable to the object on which the method is invoked (i.e. the object referred to by `this`). Other pointcut functions can be used to access information such as the arguments of a function and its return value. Furthermore, the object `tjp` (of class `JoinPoint`) allows the programmer to retrieve context information from within the advice code.

A different type of advice is that represented by introductions, which are used to augment data structures. For instance, the following code uses a slice element to add data members and methods to the class `Dialog`. A method which is defined this way can access even private data members:

```
advice "Dialog" : slice class {
  Time creation;
  int isExpired() {
    return ((Time::now() - creation) > 3600) ? 1:0;
  }
};
```

The aspect is the language construction in which all these elements are combined for the implementation of modularized cross-cutting concerns. In terms of syntaxis, an aspect is very similar to a C++ class definition. In this sense, aspects can have data members and methods, and can inherit from classes and even other aspects. The code below shows an aspect which implements an "expiration" concern for the class `Dialog`. The functionality consists of preventing the modification of any dialog window which has already

expired. In an around advice, the original join point code can be executed by calling `tjp->proceed()`:

```
aspect Expiration {
  pointcut resetTime() = construction("Dialog") ||
    execution("void Dialog::show(...)");

  advice resetTime() && that(d) : after(Dialog &d) {
    d.creation = Time::now();
  }
  advice execution("void Dialog::set%(...)")&&that(d)
    : around(Dialog &d) {
    if ( !d.isExpired() )
      tjp->proceed();
  }
};
```

CASE STUDY: THE SEQUENTIAL 2D-FFT

The Fast Fourier Transform (FFT) is used to produce frequency analysis of discrete signals in a wide range of application domains: image analysis, signal processing, speech recognition, astronomy, etc. The processing of a vector of N elements (i.e. complex numbers) involves O(N logN) operations. The FFT of a matrix (called 2D-FFT) consists of using FFT to transform each column, and then uses the result to transform each row (again using FFT).

This section describes a C++ code for the sequential processing of a stream of complex matrices using the 2D-FFT. The problem has been broken down into four main activities (sensor, FFT on columns, FFT on rows, writer) which are coupled following a pipeline scheme, so that each activity consumes the result of the previous one. The class diagram is shown in figure 1.

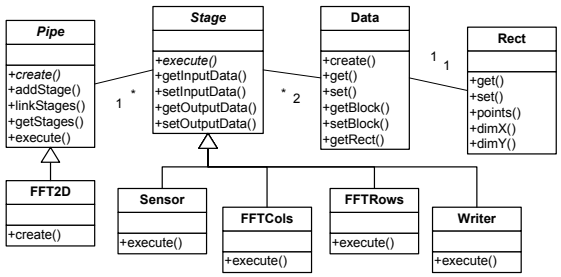


Figure 1: 2D-FFT class diagram

The class `FFT2D` models the problem as a sequential pipeline. The activities inherit from `Stage`. `Data` and `Rect` represent, respectively, complex matrices and rectangular regions. The implementation of some of the methods is described below:

```
int main(int argc, char **argv) {
  Pipe *pipe = new FFT2D;
  pipe->create();
  pipe->execute(NUM_MATS, pipe->getStages());
  delete pipe;
}

void FFT2D::create() {
  Stage *s1 = new Sensor(1);
  Stage *s2 = new FFTCols(2);
  ...
  Rect r(1,DIMX,1,DIMY);
  linkStages(s1,s2,&r);
  ...
  addStage(s1);
  ...
}
```

```
void Pipe::linkStages(Stage *s1,Stage *s2,Rect *r) {
  Data *d = new Data(r);
  s1->setOutputData(d);
  s2->setInputData(d);
  ...
}

void Pipe::execute(int iters,vector<Stage *> *stgs) {
  ...
  for (int i=0; i<iters; i++)
    for (int s=0; s<stgs->size(); s++) {
      stg = (*stgs)[s];
      stg->execute(&(stg->rect));
    }
}

void FFTCols::execute(Rect *r) {
  ...
  if ( iter == 0 )
    buffer = new Data(r);

  getInputData()->getBlock(buffer);  //input to buffer

  for (int x = r->x0; x<= r->x1; x++) {
    ...  // FFT on column x using buffer
  }
  getOutputData()->setBlock(buffer); //buffer tooutput
}
```

Each stage is associated with two `Data` objects, one of them being used as input and the other as output. Each time the stage is executed, the input is consulted so that its value can be used in the computation. The connection between two stages is implemented by sharing the same `Data` instance. This way, the i^{th} stage writes the result in the object which is read by the $(i+1)^{th}$ stage. In the class `FFTCols`, the creation of the intermediate data buffer is carried out only in the first iteration. The computation of the one-dimensional FFT entails a series of swapping and floating-point operations on the elements of the vector.

Compared to a classical implementation, the solution presented here offers some additional elements which are atypical in sequential scientific programs. They have been considered specifically with the aim of facilitating the parallelism integration by means of an AO language. Let us enumerate some of the elements:

- The degree of decomposition in terms of methods, interfaces and classes is high, therefore providing a rich set of join points to be intercepted.
- Activities are represented by different classes in the system, making the potential use of active elements easier (threads, distributed objects, etc.).
- Stages have to be linked explicitly and the linkage is based on sharing an object of class `Data`. This can be used as the basis for communicating active elements.
- Additional arguments in method calls are especially useful in two cases: for indicating the list of stages in the computation, and for setting the stage iteration range through an object of class `Rect`. The arguments can be altered by the aspect code.

Regarding the parallel execution, the solution can exploit both task and data parallelism. On the one hand, the pipeline can run its stages concurrently, which means that up to four matrices can be processed simultaneously. On the other hand, the stages

computing the FFT, which are the ones with higher computational cost, can divide the matrix into several blocks for an "embarrassingly" parallel computation. The following aspect establishes the degree of parallelism by setting both the type of data distribution and the number of blocks for every stage. This aspect declaration can be reused regardless of the parallelism implementation:

```
aspect Parallelism {
  advice "Stage" : slice class {
    int blocks;        // Number of blocks
    int distribution;  // Type of data distribution
  };

  advice construction("Stage") && that(s)
    : after(Stage &s){
    switch (s.id) {
      case 1:s.blocks=1; break;
      case 2:s.blocks=4;s.distribution=BY_COLS; break;
      case 3:s.blocks=4;s.distribution=BY_ROWS; break;
      case 4:s.blocks=1; break;
    }
  }
};
```

MULTITHREAD-BASED DESIGN

The execution of multiple threads is a way to exploit parallel hardware, as the operating system is able to run each thread on a different processor. Thread interactions can be based on variables which are allocated to a global memory space. Synchronization mechanisms are required to ensure the consistency of the shared data.

We have based the multithread implementation on the Adaptive Communication Environment ACE (Schmidt 2002), an object-oriented framework for the efficient development of concurrent communication software. The portability of ACE enables the same multithread code to be run on top of different interfaces such as POSIX PThreads, Solaris threads and Win32 threads.

The application described here uses groups of threads to execute the stages in parallel (one group per stage). The group size is indicated by the attribute `blocks` introduced in `Stage` by the aspect `Parallelism`, as stated at the end of the previous section.

Stage Connection

In the sequential version, the stages were connected by sharing a single instance of `Data`. This is a shortcoming in the parallel version since a stage running slower than the others will block the preceding ones, which will not be able to write in their output `Data` objects. A way to increase the performance is to link the stages using a buffer which supports the storage of multiple data elements. Figure 2 illustrates the idea. Four threads are running the code of `FFTCols`. Each time a result is produced, it is placed into a FIFO buffer from which the next stage (`FFTRows`, using four threads as well) retrieves its input data.

A `Fifo` class with the typical `get()/put()` methods is used for the implementation of the stage connections. As the `Fifo` objects will be accessed by different threads, synchronization mechanisms have to be considered. In our approach, the synchronization concern is developed in a separate aspect to be woven into the class `Fifo`:

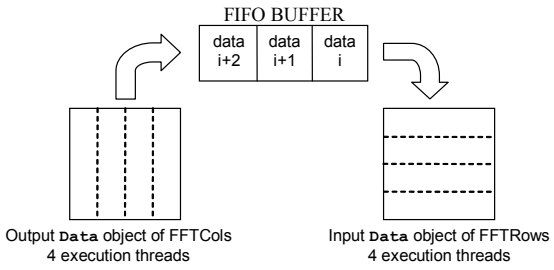


Figure 2: Stage connection scheme

```
aspect SynchFifo {
  advice "Fifo" : slice class {
    ACE_Thread_Mutex *mutex;
    ACE_Condition<ACE_Thread_Mutex> *notEmpty;
    ACE_Condition<ACE_Thread_Mutex> *notFull;
  };

  advice execution("void Fifo::get(...)") && that(f) :
    around(Fifo &f) {
    f.mutex->acquire();   // mutex acquired
    while (f.size() == 0)
      f.notEmpty->wait(); // Blocks if buffer is empty
    tjp->proceed();       // Element removal
    f.notFull->signal();  // The buffer has free space
    f.mutex->release();   // mutex released
  }
  ...  // advice on Fifo::put()
};
```

The second advice introduces code before and after the `get()` operation in order to control the removal of data. The body of the advice is accessed in mutual exclusion. If the `Fifo` buffer is empty, the thread is blocked in the `notEmpty` condition variable, waiting for any other thread to insert new data. Otherwise, a call to `tjp->proceed()` executes the code of `get()`. Then, we are sure that the buffer is not full, and so the variable `notFull` can be signaled, which will possibly wake up other threads which may be blocked in this variable because they can not complete a `put()` operation.

In order to integrate the new connection mechanism, the following advice is used to intercept the method `linkStages()`. Instead of sharing a single `Data` instance, adjacent stages will share a synchronized `Fifo` object.

```
advice "Stage" : slice class {
  Fifo *inputFifo;
  Fifo *outputFifo;

  ACE_Barrier *inputBarrier;
  ACE_Barrier *outputBarrier;
};

advice execution("void Pipe::linkStages(...)") &&
  args(s1,s2,r) && that(p)
  : after(Stage *s1,Stage *s2,Rect *r,Pipe &p) {
  ...
  Fifo *f = new Fifo(MAX_FIFO_SIZE);
  s1->outputFifo = f;
  s2->inputFifo = f;   // s1 and s2 share a FIFO
  ...
}
```

Thread Creation

The creation of threads is considered in the following advice, which affects the method `execute()` of `Pipe`:

```
advice execution("void Pipe::execute(...)") && that(p)
  && !cflow(execution("void *work(...)"))
  :  around(Pipe &p) {
  ...
  Vector<Rect> vr;

  for (int i=0; i<p.getStages()->size(); i++) {
    Stage *s = *(p.getStages())[i]; // For each stage
    s->rect.distribute(s->distribution,s->blocks,&vr);

    for (int j=0; j<s->blocks; j++) { //For each block
      arg = new ThrArg(j,iters,&p,s,&(vr[j]));
      ACE_Thread_Manager::instance()->spawn(
        (ACE_THR_FUNC)work,arg,
        THR_NEW_LWP|THR_JOINABLE);  // Thread launched
    }
  }
  ACE_Thread_Manager::instance()->wait(); //Wait
}
```

There is no call to `tjp->proceed()`, which means that the code join point will not be executed in this context. For each stage, the iteration range represented by the attribute `rect` is partitioned. Then, the group of threads is launched. The number of threads to be started is indicated in the attribute `blocks`. Every thread receives information through an object of type `ThrArg`, which includes the block identifier, the number of matrices to process, and references to the pipeline, the stage and the new iteration range.

Parallel Execution

The function `work()` executed by the threads saves the `ThrArg` variable and executes the pipeline using, this time, a stage list of only one element. The `ACE_TSS` template implements thread specific storage, which allows each thread to manage its own copy of `arg`:

```
ACE_TSS<ThrArg> arg;
static void *work(void *argument) {
  arg->set((ThrArg *)argument);
  arg->pipeline->execute(arg->iters,&(arg->stages));
}
```

Finally, data have to be moved from the `Fifo` buffers to the input and output `Data` objects each time a stage is executed. This involves the synchronization of threads:

```
pointcut invocations() =
  execution("void Sensor::execute(...)")  ||
  execution("void FFTCols::execute(...)") ||
  execution("void FFTRows::execute(...)") ||
  execution("void Writer::execute(...)");

advice invocations() && that(s) : around(Stage &s) {
  if ((s.getInputData()!= NULL) && (arg->id == 0))
    s.inputFifo->get(s.getInputData());
  s.inputBarrier->wait();   // Waits to complete input

  Rect **pr = (Rect **)tjp->arg(0);
  *pr = &(arg->rect);       // Iteration range changed

  tjp->proceed();          // Stage execution
  s.outputBarrier->wait(); // Waits to end computation

  if ((s.getOutputData()!= NULL) && (arg->id == 0))
    s.outputFifo->put(s.getOutputData());
}
```

The pointcut `invocations()` refers to the execution of any of the stages. When the advice code is triggered, the thread with `id` zero retrieves a new matrix from the input `Fifo` buffer and updates the input `Data` object. The other threads assigned to the stage are blocked until this operation is complete. Before the call to `tjp->proceed()`, the `Rect` argument, which denotes the iteration range, is replaced with the value stored in the variable `arg`. The threads wait until the code of the stage is executed. Then all the parts of the output `Data` object has been correctly updated with new values. The thread with `id` zero is allowed to insert the result into the output `Fifo` buffer.

MPI-BASED DESIGN

MPI is a collection of routines widely used in the development of parallel programs on architectures with distributed memory including computer networks. The programming model consists of a set of processes communicating by means of message-passing.

The aspect presented in this section will transform the sequential program into an SPMD application using MPI.

Application Set-up

The following advice ensures that the message-passing environment is initialized and terminated correctly:

```
advice execution("int main(...)") && args(ac,av)
  : around(int ac, char **av) {
  MPI_Init(&ac,&av);
  tjp->proceed();
  MPI_Finalize();
}
```

Unlike the code in section 4, the group of processes of an MPI application is created in a static way when the application is started. A specific computation has to be assigned to each process in the group. So, correspondence between the process rank and the pair (stage, iteration range) is required. The association in the opposite direction is also needed. The aspect defines the methods `mpiToProblem()` and `problemToMpi()` in order to carry out these mappings. Their codes are quite simple as they use the information set by the aspect `Parallelism`.

In the multithread version, the threads accessed data using shared memory, so the parallelization aspect was mainly focused on the creation and synchronization of threads. Owing to the distributed nature of MPI, the data flow in the pipeline has to be implemented by means of message-passing. The parallelization aspect in this section will be mainly focused on communications.

Data distribution must be taken into account in order to implement efficient point-to-point communications on the stage interactions. Figure 3 depicts this scenario. The stage on the left uses four processes, each one computing the FFT (on columns). When the result is passed to the next stage, data is partitioned to send remote processes the exact data pieces they require. In

this figure, process 3 has to send specific data to processes 5, 6, 7 and 8. As can be noted, the information about the data distribution and the number of blocks of the stages is essential. New attributes and methods are introduced in some classes:

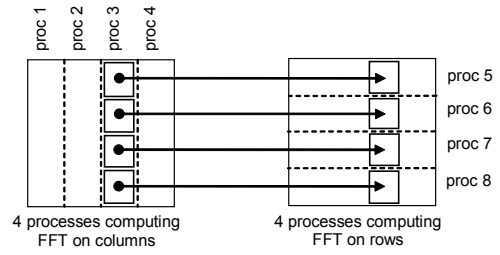


Figure 3: Communication between stages

```
advice "Stage" : slice class {
  vector<Data *> inputPartition;
  vector<Data *> outputPartition;
};

advice "Data" : slice class {
  int remote;         // Remote process
  Stage *preceding;   // Preceding stage
  Stage *next;        // Next stage

  void send(int p) { // Sends data to process p
    MPI_Send((void*)ptr,rect.points()*sizeof(complex),
      MPI_BYTE,p,MYTAG,MPI_COMM_WORLD);
  }
  void recv(int p) { // Receives data from process p
    MPI_Status status;
    MPI_Recv((void*)ptr,rect.points()*sizeof(complex),
      MPI_BYTE,p,MYTAG,MPI_COMM_WORLD,&status);
  }
};
advice execution("void Pipe::linkStages(...)") &&
  args(s1,s2,r) : after(Stage *s1,Stage *s2,Rect *r) {
  s1->getOutputData()->preceding = s1;
  s2->getInputData()->next = s2;
}
```

The meaning of the code is simple. The class `Stage` is augmented with two attributes which represent two data partitions used to receive (send) data from (to) other processes. For instance, the list `outputPartition` of the process 3 (again in figure 3) will contain four Data elements, each one being used to send a specific data piece to other process. The class `Data` is also augmented with pointers to the adjacent stages and operations to carry out the communication using MPI primitives.

The aspect declares some data members useful for the computation, such as `myrank`, `myid`, `myblock`, `mystg`, and `mystglist`. They are initialized in the following advice, which also includes the code needed to set-up the two data partition lists of the stage pointed to by `mystg` (this code has been omitted):

```
advice execution("void FFT2D::create(...)") && that(p)
  : after(Pipe &p) {
  ...
  vector<Rect> vr;
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);  // My rank
  mpiToProblem(p.getStages(),myrank,&myid,&myblock);

  mystg = p.getStageById(myid); // This is my stage
  mystglist.push_back(mystg); // List with one stage

  mystg->rect.distribute(mystg->distribution,
    mystg->blocks,&vr);
  myrect.set(&(vr[myblock])); //My iteration sub-range
  ...// inputPartition and outputPartition are filled
}
```

Parallel Execution

Finally, the following advices ensure that the parallel program will be executed correctly:

```
advice execution("void Pipe::execute(...)") :before {
  vector<Stage *> **stglist =
    (vector<Stage *> **)(tjp->arg(1));
  *stglist = &mystglist;    // Replaces the stage list
}

advice invocations() && that(stg) : around(Stage &stg)
{
  Data *d;
  // Receives data from other processes
  for (i=0; i<stg.inputPartition.size(); i++) {
    d = stg.inputPartition[i];
    d->recv(d->remote);
    stg.getInputData()->setBlock(d);
  }

  Rect **pr = (Rect **)(tjp->arg(0));
  *pr = &(myrect);  // Iteration range is changed

  tjp->proceed();   // Stage code execution

  // Sends the result to other processes
  for (i=0; i<stg.outputPartition.size(); i++) {
    d = stg.outputPartition[i];
    stg.getOutputData()->getBlock(d);
    d->send(d->remote);
  }
}
```

The first advice is needed to replace the original list of stages, which is the second argument of `execute()`, with another list containing one stage only. This way, the active element (i.e. the process) will address the execution of a single stage, instead of the complete pipeline.

The second advice uses the pointcut `invocations()` which was described in the previous section. The advice code is triggered when `execute()` is called on any of the subclasses of `Stage`. First, the process has to receive new data. More specifically, every object of `inputPartition` receives data from the corresponding remote process. These values are used to update the input `Data` object. Before the stage code is executed, the iteration range is restricted using the variable `myrect`. When the result is computed, it is sent to the processes associated with the next stage. This is done using the elements of the attribute `outputPartition`.

CONCLUSIONS

The parallelization of sequential scientific programs may entail significant changes in the code for tackling the creation, communication and synchronization of the computational tasks. In addition, the algorithm itself may be sufficiently different in the parallel version. Once these changes are applied, the resulting code probably suffers from code-tangling problems because the parallelism concerns obscure the numerical computation. Therefore, the applications become more difficult to maintain. This paper discuses the use of AOP to improve the management of the code parallelization. The benefits of the approach, as mentioned in section 1, can be expressed in terms of code modularization and reuse. It can be very difficult

to determine a theoretical result on the advantages and disadvantages of this approach. The code parallelization is traditionally done ad-hoc, so the process may vary significantly from one application to another. Thus, our conclusions will be based on the case study presented in this paper.

We consider the results to be quite promising. The complexity of the case study is moderated as it deals with the efficient integration of both task and data parallelism. The scientific core that calculates the 2D-FFT sequentially was successfully adapted to two distinct parallel scenarios (multithreading and MPI) which are characterized by quite different programming models. The only mechanism used in the code parallelization was aspect weaving (based on AspectC++, in this case). As a result, the new parallel applications have the parallelism concerns modularized into one or several aspects, and thereby code-tangling is reduced. Moreover, the same scientific core can be reused in different applications which can exploit very different parallelism models.

Regarding the development of the sequential core, the mechanisms that allowed the core to be parallelized using aspects were not difficult to implement. Basically, we have considered a good decomposition in terms of classes and interfaces, and we have included some elements, such as the iteration range of the stage, as additional arguments in some methods. Although this paper is focused on a particular case study, we are currently applying this approach to other numerical applications with more complex parallelism patterns successfully.

REFERENCES

Briham, E.O. 1988. *The Fast Fourier Transform and Its Applications*. Prentice-Hall International.

Díaz, M. et al. 2005. "An Aspect-Oriented Framework for Scientific Component Development". In *Proc. of the 13th Euromicro Conference on Parallel, Distributed and Network-based Processing PDP05* (Lugano, Switzerland). IEEE. 290-296.

Harbulot, B. and Gurd, J. 2004. "Using AspectJ to Separate Concerns in Parallel Scientific Java Code". In *Proc. of the 3rd International Conference on Aspect-Oriented Software Development AOSD04* (Lancaster, UK). ACM. 122-131.

Harbulot, B. and Gurd, J. 2006. "A Join Point for Loops in AspectJ". In *Proc. of the 5th International Conference on Aspect-Oriented Software Development AOSD06* (Bonn, Germany). ACM. 63-74.

Kiczales, G. et al. 1997. "Aspect-Oriented Programming". In *Proc. of the European Conference on Object-Oriented Programming ECOOP97* (Jyvskyl, Finland). Springer-Verlag. 220-242.

Kiczales, G. et al. 2001. "An Overview of AspectJ". In *Proc. of the European Conference on Object-Oriented Programming ECOOP01* (Budapest, Hungary). Springer-Verlag. 327-353.

Schmidt, D. and Huston, S. 2002. *C++ Network Programming: Mastering Complexity with ACE and Patterns*. Addison-Wesley.

Sobral, J.L. 2006. "Incrementally Developing Parallel Applications with AspectJ". In *Procs. of the 20th International Parallel & Distributed Processing Symposium IPDPS06* (Rodhes, Greece). IEEE.

Spinczyk, O. et al. 2002. "AspectC++: An Aspect-Oriented Extension to C++". In *Proc. of the 40th International Conference on Technology of Object-Oriented Languages and Systems TOOLS02* (Sydney, Australia). Australian Computer Society. 53-60.

AUTHOR BIOGRAPHIES

MANUEL DÍAZ. He received his M.S. and Ph.D. degree in Computer Science from the University of Málaga in 1990 and 1995, respectively. At present he is an Associate Professor in the Department of "Lenguajes y Ciencias de la Computación" (LCC) of the University of Málaga. He has worked in the areas of distributed and parallel programming and in real-time systems.

SERGIO ROMERO. He received his degree in Computer Science from the University of Málaga in 2003. At present he is a PhD student in the University of Málaga, where he works on the use of high-level software technologies, e.g. component- and aspect-based approaches, for the development of parallel and distributed numerical applications.

BARTOLOMÉ RUBIO. He received his M.S. and Ph.D. degree in Computer Science from the University of Málaga in 1990 and 1998, respectively. At present he is an Associate Professor in the Department of LCC of the University of Málaga. He has worked in the areas of distributed and parallel programming and coordination models and languages.

ENRIQUE SOLER. He received his M.S. and Ph.D. degree in Computer Science from the University of Málaga in 1990 and 2001, respectively. At present he is an Associate Professor in the Department of LCC of the University of Málaga. He has worked in the areas of distributed and parallel programming, especially in the context of high-performance scientific computing.

JOSÉ M. TROYA. He received his M.S. and Ph.D. degrees in Computer Science from the University Complutense of Madrid in 1975 and 1980, respectively. He has been a Full Professor at the Department of LCC of the University of Málaga. He has worked on parallel algorithms for optimization problems and on software engineering for distributed systems. He has been the head of the Software Engineering Group since its foundation in 1990.

Pragmatics of Virtual Machines for High-Performance Computing: A Quantitative Study of Basic Overheads

Cam Macdonell and Paul Lu
Dept. of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
Email: {cam|paullu}@cs.ualberta.ca

Abstract— **Heterogeneous administrative domains, operating systems (OS), and libraries make it difficult for computational scientists to fully utilize metacomputers and grids. Dealing with the presence or absence of features on, say, different clusters, adds complexity. How can a user or high-performance computing (HPC) application abstract out the heterogeneity?**

One possible solution is to use a virtual machine (VM) environment that supports guest operating systems and virtual disks. But, over the decades, VMs have sometimes suffered from performance overheads and limited platforms on which they can run. Through a simple quantitative study, we show that recent improvements in software and hardware support reduces the overheads for HPC applications (e.g., GROMACS, BLAST, HMMer) to under 6% for compute-intensive jobs, but 9.7% or higher for more I/O-intensive jobs, on our x86-based platform. We also argue for qualitative and pragmatic benefits of using VMs for HPC, including ease of deployment, improved functionality, and the ability to run jobs on more systems than would normally be accessible. While not perfect, VMs are emerging as a pragmatic tool in HPC.

Keywords— **metacomputing, grid computing, virtual machine (VM), bioinformatics, GROMACS (molecular dynamics simulation), benchmarking, file systems**

I. Introduction

Heterogeneity, in various forms, is often a pragmatic barrier to users taking advantage of different computer systems for a high-performance computing (HPC) workload. For example, many scientific computations in HPC consist of a set of similar jobs (sequential or parallel; we mainly focus on sequential jobs in this discussion) for a parameter sweep, such as exploring the forces between two molecules as the relative position of the molecules change [Su and Xu, 2005]. Ideally, the scientist should be able to aggregate the research group's workstations, the department's cluster, and the university's HPC consortia to run different, independent jobs from the workload. But, if the systems have different security infrastructure, run different operating systems (OS), or have different versions of software libraries, then there is a (potentially) complex process of porting and re-configuring the application and jobs for each system.

A. Background

Grid computing [Foster et al., 2002] attempts to solve some of the heterogeneity problems by mandating a class of software that needs to be installed on all systems. For example, if one installs the Grid Security Infrastructure (GSI) on all the systems, it becomes possible to support a common security model on the grid, regardless of the existing, heterogeneous security mechanisms. In essence, grid computing achieves homogeneity by defining a new, homogeneous software platform. Other projects [Lu et al., 2006], [Pinchak et al., 2003], [Anderson, 2004] partly address heterogeneity by exploiting existing software systems that are already (nearly) universally deployed (e.g., Secure Shell for security, or basic TCP/IP for client-server interactions) and minimizing the new software required for HPC workloads.

Among the remaining barriers to the mainstream use of diverse computational resources is the heterogeneity of OSes and libraries. It is not an explicit design goal of (most) grid computing nor metacomputing systems to abstract out the differences between OSes for the applications. Java, Javascript, and Flash can be described as homogeneous platforms for the World Wide Web that make it unnecessary (in theory, impossible) for applications to access the OS, thus making heterogeneous OSes less of an issue. But, existing applications have to be re-written for these platforms. And, for example, the Globus Toolkit deals with the heterogeneity of libraries (and other software or hardware) by providing tools to automate *resource discovery* (i.e., finding the platforms that have the right resources and right versions of those resources). However, resource discovery does not actually increase the number of usable systems; resource discovery locates the subset of resources that can be used.

B. Virtual Machines

How can existing, unmodified applications be supported across different platforms, regardless of what OS and libraries, or version of libraries, are available on the host system? One possible answer is through virtual machines (VM) that virtualize the physical hardware, such as VMware [Adams and Agesen, 2006] (`www.vmware.com`), Parallels (`www.parallels.com`), and, historically, IBM's System/360 VM. (Note that Parallels is the name of the commercial product and is not specifically referring to parallelism in HPC.) Unlike Java VMs, VMware (and similar systems) virtualize the hardware without changing the instruction set of the processor or the standard ways of interfacing to input/output (I/O) devices. There are a number of other approaches to virtualization, including Xen's paravirtualization [Barham et al., 2003], KVM's Linux kernel-based approach [Qumranet, 2006], and the forthcoming Windows-based strategy from Microsoft. For this study, we focus on VMware because of its relative maturity and current wide availability. A comparison between different VMs and strategies is the subject of future work.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

By virtualizing the hardware, a *host server* and *host OS* can host the VM, which in turn can run a *guest OS*. On top of the guest OS, an unmodified version of the application can execute as if it was running on bare hardware with the appropriate OS. Note that we use the term "bare hardware" (as opposed to "virtualized hardware") to refer to the *combination* of hardware and a (host) OS throughout this discussion.

For example, consider an instance of the GROMACS application (a molecular dynamics (MD) simulator) [Lindahl et al., 2001] that is normally executed on an x86-based server, running a Linux 2.4-based distribution, with libraries from the year 2005. VMware allows the creation of a VM (i.e., a set of files containing the contents of virtual disks) that contains Linux 2.4, the necessary libraries, and GROMACS itself. The resulting VM can run on a host system that consists of, say, x86-based hardware and running Microsoft Windows XP. Furthermore, VMware can run the *same* VM on a variety of guest OSes, including Linux (whether 2.4-based or 2.6-based), Mac OS X, and other versions of Windows. By packaging a VM with GROMACS, Linux 2.4, and libraries, it should be possible to run multiple instances of the *same VM* on heterogeneous host OSes, with different versions of the same OS, and with different versions of their required libraries.

However, the VM-based approach does have some disadvantages:

1. It is non-trivial to create a VM for a scientific application. Packaging the OS, libraries, as well as the application itself requires more expertise (and effort) than is typical of most computational scientists.
2. Contemporary VM products, such as VMware, are limited to x86-based hardware platforms.
3. Virtualization has overheads [Adams and Agesen, 2006]. Emulating different I/O devices and dealing with the issues of privilege (in the traditional sense for OSes) results in a loss of some performance, compared to running directly on a host OS and hardware.

The goal of this paper is to evaluate virtualization overheads, in the context of HPC applications, to consider some of the pragmatic issues related to VMs, and to draw some appropriate conclusions about the advantages and disadvantages of VMs for HPC. After a set of simple, quantitative experiments, we conclude that VMs are promising tools for compute-intensive applications, and are less well-suited for I/O-intensive applications.

II. The Pragmatics of VMs

A number of arguments have been made in favour of using VMs on grids and similar environments [Figueiredo et al., 2003]. In this section, we focus on three main pragmatic reasons to consider VMs. In the next section, we consider the performance of full-sized applications.

Pragmatics 1: Virtual Appliances: Although creating a VM from scratch does require expertise, a growing trend with VMs is to create so-called *virtual appliances*. For example, a Linux 2.6-based distribution (be it Gentoo, Debian, Scientific Linux, or any other) can be packaged into a VM and distributed as a unit, analogous to shrink-wrapped consumer software. In particular, VMware now supports an online user community where dozens of pre-packaged appliances have been created and are available for download (`http://www.vmware.com/vmtn/`). Examples of appliances include self-contained mail servers, network firewalls, and distributed file systems [Closson and Lu, 2005]. And, the virtual appliances can be used with VMware's free-to-use versions of their VM, known as VMware Player (analogous to the free-to-use version of Adobe Acrobat, known as Acrobat Reader) and VMware Server (which is different than the non-free VMware ESX Server).

Due to the open-source licence of Linux and related software, users can install their applications on the VM, and the resulting appliance can also be redistributed. One can imagine GROMACS being installed on top of an existing Linux-based virtual appliance to create, say, a GROMACS appliance. Therefore, most computational scientists do not have to become experts in installing and configuring Linux; they install their software on the VM in the same way they install their applications on their Linux cluster. Or, they download a pre-made application-specific appliance.

Whether packaging one's own VM or using a pre-made appliance, the result is a system that can be used *without changes* on a variety of host OSes, regardless of what version of the OS or libraries are on the host. That is a significant reduction in the amount of heterogeneity that the computational scientist has to be concerned about.

We have built a virtual appliance of our own, the Trellis NAS Bridge Appliance (Trellis NBA or TNBA). It is a virtual appliance that provides "bridged" file access [Closson and Lu, 2005] across administrative domains using Secure Shell as the basic security mechanism [Lu et al., 2006]. In HPC, explicit stage-in/stage-out of data is common, tedious, and error-prone. Using a combination of Samba (`www.samba.org`) and technology from the Trellis Project, TNBA allows unmodified binary applications to access files (using open, read, write, and close, as per a file system) instead of via copying or file transfer. We believe that the packaging of virtual appliances with a distributed file system will provide a useful platform for HPC, especially if the performance overheads are negligible (Section III-E).

Pragmatics 2: x86 Platform: Although VMware and most contemporary VMs are limited to x86-based hardware platforms, it is such an ubiquitous platform that it is still meaningful to consider the VM-based approach. Furthermore, a virtual appliance can be run on (almost) any x86-based system running Microsoft Windows, Linux, and Mac OS X. Therefore, in some situations, the number of actual servers that are usable with the VM can grow in some ways (i.e., more OSes) while shrinking in other ways (i.e., non-x86 servers).

Pragmatics 3: Overheads: Perhaps the most worrisome aspect of VMs is the potential loss of performance. In HPC, a great deal of money and effort is spent to increase performance by a few percent. How much of an overhead does the VM incur? Why would anyone be willing to use an approach that had any additional overhead?

This paper attempts to quantify the basic overheads of the VM approach. Using the GMX benchmarks distributed with GROMACS, we measure the overhead to be less than

6% (Section III-C, Figure 3) for the more compute-intensive workloads. For more I/O-intensive workloads, we measured overheads as high as 9.7% when comparing the VM against bare hardware. Therefore, the VM-based approach may not be ideal for all scientific HPC applications. But, if one's application is similar to GROMACS, then for about a 6% reduction in performance, one can gain the benefits of a portable virtual appliance (to deal with heterogeneity) that can potentially run on many different x86-based OSes.

III. Experiments

The goal of the experiments is to answer the question: Can the current generation of VMs, as represented by VMware, be competitive enough with bare hardware to warrant consideration for throughput-oriented HPC workloads?

To measure the overheads of running scientific applications, three widely used scientific applications related to bioinformatics and biological simulation are measured on hardware and under VMware on the same platform. The benchmarks are BLAST [Altschul et al., 1990] (sequence matching in bioinformatics), HMMer [Eddy, 1998] (machine-learned pattern matching), and GROMACS [Lindahl et al., 2001] (molecular dynamics simulation). These benchmarks are in wide use and often run on clusters with a batch scheduler. Also, GROMACS and HMMer have both been included in the recently released SPEC CPU 2006 benchmarks.

All real-time data points are the averages of five runs, as measured using `gettimeofday()`, where the observed standard deviation of times is very low. There is some concern about the use of `gettimeofday()` inside VMs, so we also experimented with using the timestamps on network `ping` packets to measure real time [VMware, Inc., 2006]. We found differences of less than 1% in the timings between `gettimeofday()` and `ping` for our runs.

Our test machines are dual Opteron (model 248, running at 2.2 GHz) Linux servers with 4 GB of RAM and a 250 GB disk. The virtual environment is run under VMware Server version 1.0.1, which is free-to-use after registering. Note that we did *not* use the higher-performance VMware ESX Server, which has an associated licencing fee. Benchmarks using VMware ESX Server would be of interest, but for practical reasons, those experiments are left for future work.

Virtual machines were configured with 2 GB of RAM, which is sufficient for all of the non-I/O-related needs of the benchmarks. For our experiments, the memory allocated to a VM is less than the total physical memory of the host hardware. We used 2 GB of RAM in the VM even though the server had 4 GB of RAM since our applications did not require all of the memory, and to eliminate any issues related to overcommitting memory and paging. In general, the amount of useful memory for the VM will be less than the total physical memory, which is one of the trade-offs in the VM-based approach.

Except where it is noted in the results, the virtual machines were configured to use two processors. The operating system on the servers ("host OS") is Scientific Linux 4.4 (Linux Kernel 2.6). Our "guest OS" inside the virtual machines is Gentoo Linux 2006.1 (Linux Kernel 2.6). Our experience is that there are no significant *performance* differences between the two Linux distributions; we chose both Scientific Linux and Gentoo to emphasize how VMs allow for different guest and host OSes. Each virtual machine has two 16 GB virtual disks. Growable disks are used for all experiments, but we also report on performance with preallocated disks in selected situations (Section III-D). One disk is used for system data and the second is used exclusively to run the experiments.

In the first part of our study, we compare applications running under a virtual machine, VMware Server, versus running on bare hardware. In this experiment, both the host and guest OSes are 64-bit versions. In the second part of our study, the GROMACS and BLAST applications run within a virtual machine. The virtual machine in this experiment is the current version of our Trellis NAS Bridge Appliance (TNBA) (with the Trellis File System) (Section II), which is packaged with a 32-bit version of Gentoo as the guest OS. The ability to run a 32-bit guest OS on a 64-bit host OS is another example of how VMs can abstract heterogeneity (Section III-F). We ran benchmarks with the data located outside the virtual machine on a different machine on the local network and brought in "on demand". The machines are connected with gigabit Ethernet across a switch. We compare two techniques for accessing the remote data: the Trellis File System and stage-in/stage-out.

A. BLAST

BLAST (Basic Local Alignment Search Tool) is a widely used tool for finding similar nucleotide or protein sequences. A typical input is a single sequence which is compared against a database of known sequences. The most statistically similar sequences are computed and returned. The BLAST tests were taken from two sources: (1) the Bioinformatics Benchmark System (BBS) benchmark (version 3) and (2) the Pathway Analyst project here at the University of Alberta. For our experiments, we used BLAST version 2.2.15.

The BBS BLAST benchmarks were used in the first part of our study, along with the FASTA NR and PATNT databases from NCBI dated December 12, 2006. The BBS benchmark tests three different programs in the BLAST suite. Using `vmstat`, we did a rough characterization of the I/O intensity of the different programs and found that they required averages of 8,600 blocks per second (bps), 1,016 bps, and 219 bps (of 4 kilobyte blocks) of I/O for the single CPU versions of *blastn*, *blastx* and *tblastx*, respectively. The algorithmic differences between these programs are beyond the scope of this paper, but each varies in its input, comparison method, and measured I/O intensity.

The Pathway Analyst BLAST test (Table I) was used in the second part of our study. For the Pathway Analyst test, the proteome (all proteins) of *E. coli* was compared against 43 organisms, including itself, from the KEGG dataset downloaded on September 28, 2006. In total, 18,841 sequences were searched for against 289,770 sequences in the database. The program `blastp` is used in the Pathway Analyst benchmark. `blastp` compares an amino acid query sequence against a protein sequence dataset.

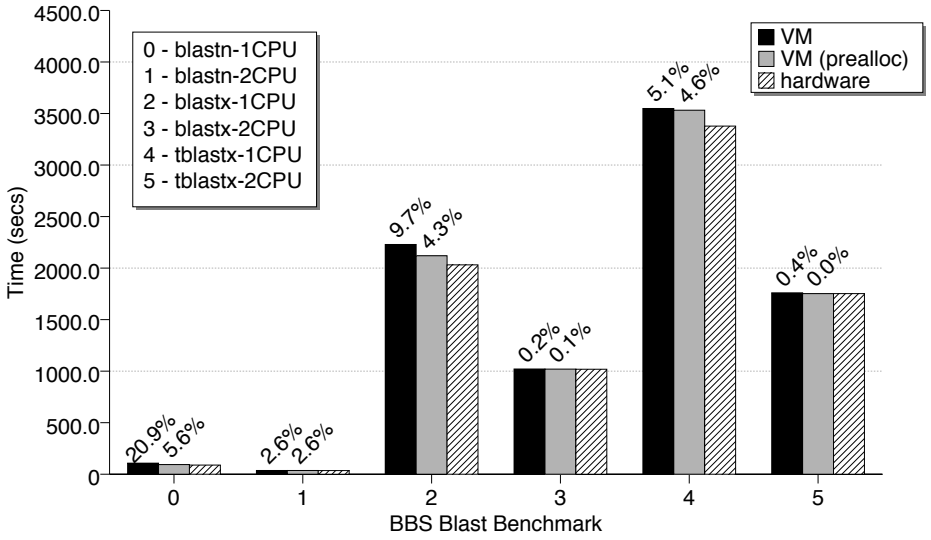


Fig. 1. Performance of the BBS BLAST Benchmark under VMware Server and on hardware. Percentage overhead versus hardware shown above the bar: VM with growable disks and VM with preallocated disks.

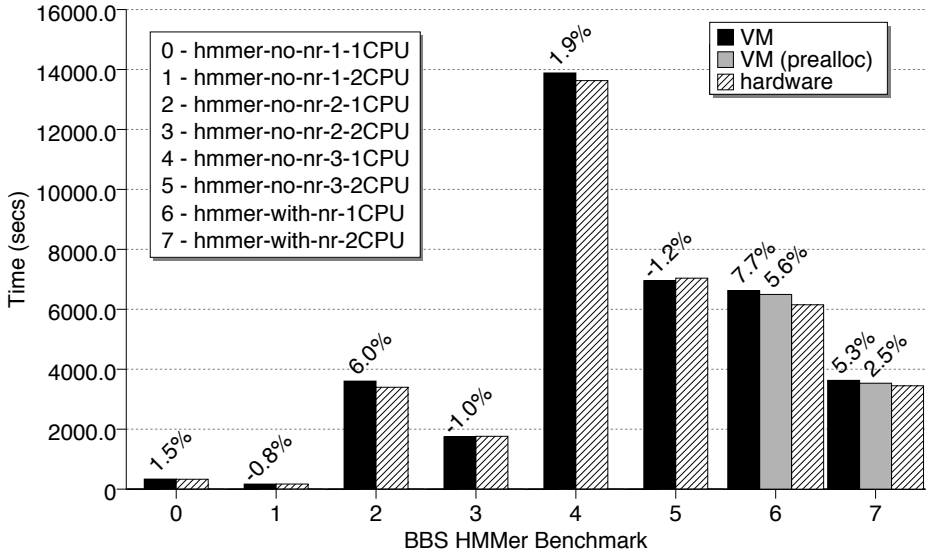


Fig. 2. Performance of the BBS HMMer Benchmark under VMware Server and on hardware. Percentage overhead versus hardware shown above the bar: VM with growable disks and VM with preallocated disks. Datapoints are without (0 to 5) and with (6 and 7) the NR database.

B. HMMer

HMMer (pronounced "Hammer") is a profile hidden Markov model (HMM) implementation that generates statistical descriptions of sequence families and searches databases for similar sequences. Similar to our BLAST benchmarks, the inputs for HMMer were taken from the BBS. Version 2.3.2 of HMMer was used for the tests. The HMMer benchmark includes HMM training applications and search applications. HMM training (labeled *hmmer-no-nr* in the graphs) are compute-intensive with little I/O. The HMM search applications (labeled *hmmer-with-nr* in the graph) are a database search and so are I/O-intensive, as confirmed by `vmstat`. For the HMM search, the FASTA NR database mentioned above was used as the database to search against.

C. GROMACS

GROMACS is a suite of applications used for molecular dynamics (MD) simulations. It generates the trajectories of the atoms in a molecule, in water, over a period of time, typically on the order of picoseconds or nanoseconds. For this study, we measure the runtime of the computationally intensive `mdrun` program that actually performs the simulation. GROMACS v3.2.1 was used with the FFTW v2.2.15 library. We used GROMACS v3.2.1 as it is the version used by computational biologists that we collaborate with. The newer GROMACS v3.3.1 has had issues with different versions of the GNU C compiler (`gcc`) that has kept our collaborators from using it. As input, we used the GROMACS benchmarking system *gmxbench* that consists of four molecules published by the GROMACS group. The four molecules in

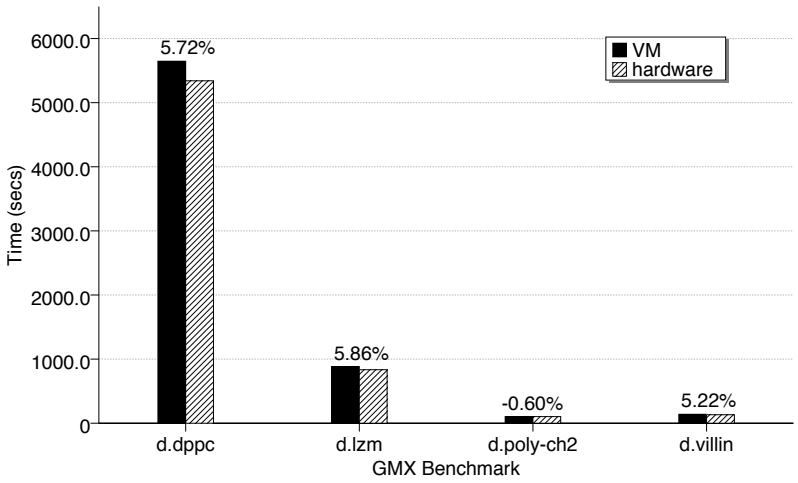


Fig. 3. Performance of GROMACS *gmxbench* under VMware Server and on hardware. Percentage overhead versus hardware shown above the bar: VM with growable disks only.

gmxbench are *d.dppc*, *d.lzm*, *d.poly-ch2*, and *d.villin*. This benchmark is used in both Sections III-D and III-E.

D. Results

Figure 1 shows the results for the BBS BLAST benchmark, with one and two processors via shared-memory parallelism. Each set of bars compare the same benchmark run under VMware versus being run on hardware. Both growable and preallocated disks are evaluated. The percentages above the bars are the percentage overheads in runtime under virtualization.

For growable disks (i.e., virtual disks that allocate storage on demand, instead of at time of creation), the overheads vary from 0.2% to up 9.7% for substantial runs, with a high of 20.9% overhead for a small dataset run. The standard deviation on the average of five runs for each data point is approximately 4%. There are data points (in most of the experiments discussed in this section) in which the VM and hardware times are almost identical, or where the VM times are nominally "faster" than on real hardware by a small amount. However, we focus on the cases where the differences are significant to better understand the worst-case scenarios. For space reasons, all BBS benchmarks are numbered, with the names given in the legend.

Whereas growable disks use less storage for sparsely populated virtual disks, preallocated disks have lower performance overheads [VMware, Inc., 2006]. Our experiments support this tuning guideline as the runtime overheads for BLAST are reduced to between 0% to 5.6% when using preallocated disks. Therefore, if storage efficiency is key, then growable disks can be used, but preallocated disks offer higher performance for BLAST.

Figure 2 shows the results of the BBS HMMer benchmarks run in a similar manner to the BLAST benchmarks. The overheads are shown to be as low as minus 1.2% with growable disks (which is within the standard deviation of the data point). The lower overheads are on the HMMer training benchmarks (i.e., without the NR database, datapoints 0 to 5). The two sets of bars to the right of the graph (labeled 6 and 7) are the HMM searches with overheads as high as 7.7%, with growable disks. These datapoints have higher I/O rates (measured via `vmstat` to be up to 335 bps on average) as they must search the NR database, which is 11 GB in size. With preallocated disks, the overheads drop to 5.6% and 2.5% for the single and dual CPU versions, respectively.

The results of the GROMACS *gmxbench* benchmark are shown in Figure 3. The overheads for *gmxbench* are from minus 0.60% to 5.86%. The `mdrun` program generates four output files that describe the simulated molecular dynamics of the input molecules. These files vary from 200 KB for *d.poly-ch2* to over 2 MB for *d.dppc*, which are small by current standards. As used in *gmxbench*, GROMACS is known to be compute-intensive and is representative of a large class of simulation-based applications.

E. Using Trellis File System for Remote Data Access

As an example of the pragmatic benefits of packaging and virtual appliances (Pragmatics 1, Section II), the Trellis File System can be pre-installed and configured in the VM, along with the application itself. Normally, deploying a distributed file system is either prohibitively complex for most computational scientists or the file system requires privileged access to install. With the VM-based approach, the Trellis File System can be made available as a virtual appliance. The scientist can then co-install their application with the appliance and use it. Running the VM-based appliance and application on a compute node requires no special privilege, since all of the privileged steps are encapsulated inside the VM.

Efficient remote data access is also important in being able to leverage virtualization in metacomputers. Two techniques for accessing remote data were measured in this part of the study. The first is the common practice of using scripts to stage the data in and out of a compute node (in our case the VM). The second is using the Trellis File System to access the data. The Trellis File System accesses the remote nodes and exports a file system interface through a Samba server running inside the virtual machine. For this part of our study, we used the *gmxbench* benchmark and the BLAST search from the Pathway Analyst project (Table I).

Benchmark	Trellis NAS Bridge Appliance	Stage-in/Stage-out		
	Total (% overhead vs. stage-in/out)	Total =	computation +	`scp`
GROMACS d.dppc	4,412.7 (2.6%)	4,307.1	4,300.8	6.3
d.lzm	607.35 (0.6%)	603.7	601.2	2.5
d.poly-ch2	102.7 (1.2%)	101.4	99.4	2.0
d.villin	89.7 (2.5%)	87.5	85.9	1.6
Pathway Analyst BLAST	15,005.5 (5.5%)	14,218.5	14,182.0	36.5

TABLE I: Remote data access from within a virtual machine: Trellis NBA v. Stage-in/Stage-out for GROMACS and BLAST (times are in seconds).

For both remote access methods, the benchmark application is running within the VM and the data is stored on a different node in the same cluster.

There is a row for each of the GROMACS *gmxbench* molecules and the last row is for the Pathway Analyst BLAST run (Table I). The second column is real time in seconds of the benchmark using Trellis NBA to access the remote data and store the results back to the home node. The GROMACS and BLAST applications are unmodified and use the Trellis File System via pre-configured mount points inside the VM. The percentage overhead of Trellis NBA versus stage-in/stage-out is in parentheses. For stage-in/stage-out, the total times are given as well as a breakdown of the time between computation (`mdrun` for GROMACS and `blastp` for BLAST) and explicit data movement via Secure Copy (`scp`).

The additional overheads of TNBA vary from 0.6% to 2.6% for GROMACS and the overhead is 5.5% for the BLAST job from Pathway Analyst. The overheads are due to the use of Samba and Trellis (i.e., extra software layers) within the virtual appliance. For future work, we plan on optimizing the software and improving performance. Currently, for a trade-off of less than 6% in additional overhead, the Trellis NAS Bridge Appliance can provide a distributed file system that does not require any special privilege to install and use, other than installing VMware Server itself.

F. 32-bit vs. 64-bit Assembler Optimizations

When running the GROMACS application in Section III-E (Table I), we observed that the runtimes were lower than both the hardware and VM times from Section III-D (e.g., 4,412.7 seconds in Table I versus 5,340 seconds, which is the left-most *hardware* bar in Figure 3 for *d.dppc*). After some investigation, we realized that running in the 32-bit Trellis NAS Bridge Appliance VM (Table I) was the cause of the difference. The VM hid the fact that the physical processor was an 64-bit Opteron (x86-64) and so the guest OS and GROMACS detected a 32-bit i686 processor. The GROMACS configuration then compiled highly-optimized i686 assembler loops into the GROMACS application. These optimized assembler loops resulted in the much lower runtime. The version of GROMACS (version 3.2.1) we were running did not support the same optimizations under x86-64, so the GROMACS in Section III-D was slower. After tracking down the cause, we also found that the i686 optimizations could not be compiled on the host 64-bit OS. More recent versions of GROMACS (e.g., version 3.3.1) do have the optimized loops for 64-bit OSes.

A full comparison of 32-bit versus 64-bit issues is beyond the scope of this paper. And, this anomaly is likely transitory. But, as discussed above, the ability to run a 32-bit guest OS on a 64-bit host OS is another example of how VMs can abstract heterogeneity.

IV. Related Work and Comments

Other researchers have studied the overheads of VMs [Adams and Agesen, 2006] and the strategy of using VMs to encapsulate HPC jobs [Figueiredo et al., 2003], [Santhanam et al., 2005]. Our study extends and updates those studies by choosing full-sized applications (e.g., GROMACS, BLAST, HMMer) that are in wide use in our HPC community, and our research group (e.g., the Proteome and Pathway Analyst projects in our department use BLAST and HMMer), instead of relying on industry-standard (but generic) benchmarks such as SPEC [Adams and Agesen, 2006].

The use of virtual machines in distributed environments has been studied before. As part of the In-VIGO system, Figueiredo *et al.* [Figueiredo et al., 2003] examine using virtual machines for distributed computation within a grid framework. They also use VMware. To measure overheads, the authors used two compute-intensive SPEC benchmarks. They report overheads of 1-4% which are lower but consistent with our findings for compute-intensive applications. Their paper did not analyze I/O-intensive applications, but they did measure the impact of other factors such as competing processes. Zhao *et al.* [Zhao et al., 2004], also part of In-VIGO, designed the grid virtual file system (GVFS), a distributed file system for efficiently moving virtual machines across the wide-area network and providing data access within VMs. Network latency for VM images and application data are not separated in their experiments, so the overheads specifically from virtualization are not clear. The conclusions in both papers from In-VIGO are similar to ours: the overheads of virtual machines are acceptable for the benefit they provide in abstracting heterogeneous environments.

Santhanam *et al.* [Santhanam et al., 2005] examine running Condor computations within a virtual environment. The focus is similar to our Pragmatics 1 (Section II) in using VMs as *sandboxes* for the grid. They focus on running computations in four different configurations of Xen virtual machines. The variations involve the location of data and network connectivity of the VM. Their experiments measure data-intensive microbenchmarks involving thousands of concurrent reads and network operations. They characterize the impact that different data access methods, such as

remote I/O and whole-file caching, have on performance. Their baseline for comparison of the VMs is the Condor Vanilla and Standard Universes [Litzkow et al., 1988]. Condor runs batch jobs in distributed, heterogeneous environments. The Condor Vanilla Universe is the closest to our hardware experiments. As expected, their experiments reveal I/0 overheads under virtualization. However, due to their exclusive use of microbenchmarks, it is not clear what overheads can be expected for applications.

Our quantitative evaluation uses full-sized HPC applications instead of microbenchmarks. Moreover, our use of the commercial, widely-available VMware, instead of the more research-oriented Condor and Xen combination [Santhanam et al., 2005], is arguably more applicable for our production environments.

V. Concluding Remarks

Although virtual machines have been around for decades, recent developments in VMs for x86-based platforms have re-opened the VM debate. A variety of VMs are now available commercially (e.g., VMware, Parallels) and as open-source software (with commercial support) (e.g., Linux KVM, Xen). Pragmatically, VMs are a convenient way to package and deploy scientific applications across heterogeneous system. For example, applications can be packaged with their required libraries and support programs, including (perhaps) a distributed file system (e.g., Trellis NAS Bridge Appliance) that would otherwise be difficult or impossible to install without special privilege.

However, an important concern about VMs is whether or not the associated overheads are too onerous. In our simple, quantitative study, we show that the overheads for a compute-intensive application, such as GROMACS, can be under 6%. For more I/O-intensive applications (e.g., BLAST, HMMer with NR database), the overheads can be as high as 9.7%. The overheads will vary depending on the application itself, which is why we chose to do a contemporary evaluation of well-known scientific applications (i.e., BLAST, GROMACS, HMMer) using a modern VM (i.e., VMware). A performance comparison of different VMs and with a broader range of sequential and parallel applications is the subject of future work.

There will always be reasons to not give up any performance at all. But, for the convenience and other benefits of VMs, there may be other situations where the cost-performance trade-off is worth re-visiting.

VI. Acknowledgments

This research is supported by the Natural Science and Engineering Research Council of Canada (NSERC), the Alberta Prion Research Institute (APRI), the Alberta Ingenuity Centre for Machine Learning (AICML), Alberta Ingenuity, SGI, the Alberta Science and Research Authority (ASRA), and Sun Microsystems. Thank you to the Trellis Project team; Jordan Patterson and the Proteome/Pathway Analyst teams. Thank you to Hemant Gaidhani, Priti Mishra, and VMware, Inc. for permission to report the VMware benchmarks.

References

[Adams and Agesen, 2006] Adams, K. and Agesen, O. (2006). A Comparison of Software and Hardware Techniques for x86 Virtualiztion. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, U.S.A.

[Altschul et al., 1990] Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *J Mol Biol*, 215(3):403–10.

[Anderson, 2004] Anderson, D. (2004). BOINC: A System for Public-Resource Computing and Storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, Washington, DC, USA.

[Barham et al., 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, New York, NY, USA.

[Closson and Lu, 2005] Closson, M. and Lu, P. (2005). Bridging Local and Wide Area Networks for Overlay Distributed File Systems. In *Proc. 2nd Workshop on Real, Large Distributed Systems (WORLDS '05)*, pages 49–54, San Francisco, California, U.S.A.

[Eddy, 1998] Eddy, S. (1998). Profile Hidden Markov Models. *Bioinformatics*, 14:755–763.

[Figueiredo et al., 2003] Figueiredo, R., Dinda, P., and Fortes, J. (2003). A Case For Grid Computing On Virtual Machines. In *Proc. 23rd International Conference on Distributed Computing Systems (ICDCS)*, page 550, Washington, DC, USA. IEEE Computer Society.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum.

[Lindahl et al., 2001] Lindahl, E., Hess, B., and van der Spoel, D. (2001). GROMACS 3.0: A package for molecular simulation and trajectory analysis. *J. Mol. Mod.*, 7:306–317.

[Litzkow et al., 1988] Litzkow, M., Livny, M., and Mutka, M. (1988). Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*.

[Lu et al., 2006] Lu, P., Closson, M., Macdonell, C., Nalos, P., Ngo, D., Kan, M., and Lee, M. (2006). The Trellis Security Infrastructure for Overlay Metacomputers and Bridged Distributed File Systems. *Journal of Parallel and Distributed Computing*, 66(9):1181–1188.

[Pinchak et al., 2003] Pinchak, C., Lu, P., Schaeffer, J., and Goldenberg, M. (2003). The Canadian Internetworked Scientific Supercomputer. In *17th International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 193–199, Sherbrooke, Quebec, Canada.

[Qumranet, 2006] Qumranet (2006). KVM: Kernel-based Virtualization Driver (White Paper). `http://www.qumranet.com/wp/kvm_wp.pdf`.

[Santhanam et al., 2005] Santhanam, S., Elango, P., Arpaci-Dusseau, A., and Livny, M. (2005). Deploying Virtual Machines as Sandboxes for the Grid. In *Second Workshop on Real, Large Distributed Systems (WORLDS 2005)*, San Francisco, CA.

[Su and Xu, 2005] Su, Z. and Xu, Y. (2005). *Ab initio* study of chiral recognition in the propylene imine-hydrogen peroxide complex. *Phys. Chem. Chem. Phys.*, 7:2554–2560.

[VMware, Inc., 2006] VMware, Inc. (2006). Performance Benchmarking Guidelines for VMware Workstation 5.5 (White Paper). `http://www.vmware.com/pdf/WS55_Benchmarking_Guidelines.pdf`.

[Zhao et al., 2004] Zhao, M., Zhang, J., and Figueiredo, R. (2004). Distributed File System Support for Virtual Machines in Grid Computing. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 202–211, Washington, DC, USA. IEEE Computer Society.

VII. Author Biographies

CAM MACDONELL is a Ph.D. student in the Dept. of Computing Science, University of Alberta, Edmonton, Alberta, Canada. His research interests are high-performance computing and operating systems.

PAUL LU is an Associate Professor of Computing Science, University of Alberta, Edmonton, Alberta, Canada. His research interests are high-performance computing, operating systems, and bioinformatics.

Multi-RAID Queueing Model with Zoned Disks

Soraya Zertal[1] and Peter Harrison[2]
[1] PRiSM, Université de Versailles, 45 Av. des Etats-Unis, 78000 Versailles, France
Zertal@prism.uvsq.fr
[2] Imperial College London, South Kensington Campus, London SW7 2AZ, UK
pgh@doc.ic.ac.uk

Abstract— **A queueing model is developed for a multi-RAID storage system implemented on modern zoned disks, using fine, accurate access time functions. An extension of a previous analytical model that utilizes Fork-Join composition of M/G/1 queues, it describes zoning directly in terms of the probability distributions or moments of the model's components, such as seek time, rotational latency and data transfer time. These quantities are calculated directly using the principles of operation of the hardware. This is in contrast to estimating them from simulations and theoretical bounds, as in previous zoned disk models. The resulting multi-RAID model turns out to be accurate, when its performance predictions, characterized here by the mean of queueing and response times, are compared with simulation, and also scalable; not only for the zoned technology but also for alternate ones.**

keywords : Multi-RAID, Zoned disks, Fork-Join, M/G/1 queues, I/O modelling, Simulation.

I. Introduction

Computer applications become ever more data intensive. To satisfy their QoS requirements in terms of capacity, performance and availability, RAID (Redundant Arrays of Independent Disks) are commonly used. To describe and predict RAID performance, several analytical and simulation models have appeared in the literature since the introduction of RAID [2] but their objective is always one and only one RAID configuration per disk array. However, it has been shown that with the continous evolution of data access to disk arrays, it is necessary to provide different RAID configurations on the same array to adapt the data storage to user requirements on access time and availability. This leads to better exploitation of the storage system's space and improved performance of its usage. With the introduction of this type of multi-RAID system, none of the available RAID models is capable of describing and predicting its performance effectively.

We introduced a new analytical queueing methodology for this purpose in [6], [25] and this is still the only one addressing such a complex disk array, to our best knowledge. Any analytical performance modelling of storage systems is concerned with expressing mathematically the details of its devices' technology. This is not yet the case for our multi-RAID model since, to date, it cannot account for zoned disks. The aim of this paper is to extend our previous modelling study [8] by taking into account both the fine details of modern zoning disk technology and more complex, but more representative, access time functions. The result is an accurate and scalable multi-RAID model.

In the rest of the paper, section 2 summarizes RAID systems and modelling in this area. Section 3 details the new, analytical model, describing zoned disks and the aforementioned access time functions, whereas Section 4 describes the simulation procedures. Section 5 discusses and compares the numerical results obtained from the analytical model and the simulation. Finally, section 6 concludes the paper and suggests future research directions.

II. Overview of the RAID system and related work

A RAID storage system consists of a disk system manager and a collection (array) of independent disks. The disk system manager is a software component of the RAID controller. It receives logical requests from the multiple system users, at different rates, subdivides the data into blocks and distributes them across the disks. Consequently, for each logical request, it generates a number of physical requests and sends them to the associated disks which receive requests at different rates. Finally, the disk system manager waits for responses from each requested disk to construct the (logical) response to send to the user. The request subdivision-distribution process is performed according to the data/redundancy pattern over the disks. When various data placement schemes [2] coexist with dynamic selection of the current redundancy pattern, in order to provide storage space and access time optimizations, we obtain a dynamic Multi-RAID [24]. Requests' independent executions on such asynchronous disks lead to Fork-Join-type modelling problems, described and analysed in [8].

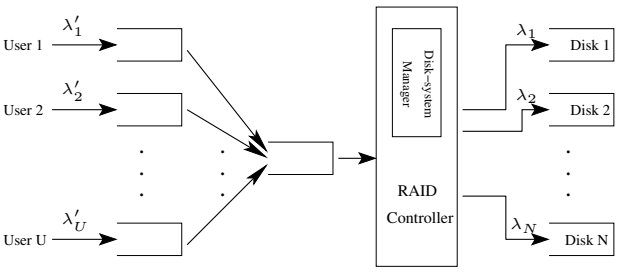


Fig. 1. Requests flow in a RAID storage system

Modelling of RAID systems is the subject of several studies. For a few examples, we can cite the approximation of access latency [11], queueing models for RAID3 and RAID5 [1], RAID queueing models in different execution modes – normal, degraded and recovery [13], [14], reconstruction and failure tolerance [5]

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

and caching and controller optimization [3], [23]. All of these models are appropriate for different RAID levels but just one at a time. Thus, none of them is appropriate for a multi-RAID system. To our knowledge, only the model presented in [8] can handle analytically all the characteristics of a multi-RAID system and can be used to predict its performance. In fact, on a multi-RAID, the unbalanced workload leads to a difference between disks' waiting times and the use of asynchronous disks introduces a significant difference between the seek times. The RAID is then presented as a collection of M/G/1 queues, one for each disk, interacting so as to account for *synchronization* between the disks. This interaction may be approximated by estimating the mean of the maximum of the response times among the disks involved in a logical access. In fact, an exact solution for this type of Fork-Join problem was obtained for Poisson arrivals and exponential service times, together with an approximation for other distributions in [6], [8]. A detailed evaluation of such approximations is the subject of [7]. However, in this multi-RAID model, it was assumed that all disk cylinders are identical and the seek time is calculated using a simple function as derived in [18].

Recently, a new disk organisation emerged called *zoned disk technology*, in which the number of sectors per track is variable. Consecutive cylinders are collected into groups, called *zones*, such that within each zone, the track capacity (number of sectors) and the transfer rate are fixed. However, these two parameters decrease from the outer to the inner zones. These disks have become very popular due to their greater storage capacitiy and transfer rate. Their average rotational latency is constant but the variable seek and transfer times necessitate more complex calculations in terms of the assumed statistical workload and disk behaviour in a storage model [16], [17].

RAID modelling relies strongly on the analysis of Fork-Join queueing networks. Until now, research has mainly been concerned with approximations, performance bounds and calculations based on simulations. In [21], [20], a performance bound is calculated for a closed Fork-Join network. In [15], the Fork-Join response time for homogeneous processes with exponential service time distributions is calculated for two processes and approximated for more, based on simulations and theoretical bounds. In [22], Fork-Join queueing systems with general service times are approximated using interpolation and in [19], the approximation of similar systems is based solely on preliminary simulations.

III. Analytical model

The entire RAID model that we address is based on a collection of M/G/1 queues with various extensions to account for the Fork-Join nature of the parallel disk accesses corresponding to a logical request. The response time of each physical request, to an individual disk, is composed of four components: the time spent waiting to start service in the disk queue (Q), the seek time (S), the rotational latency (R) and the transfer time, which we separate into two components, t and T, corresponding to transfer from the disk's cylinder to its buffer and from this buffer via the bus, respectively. The model we develop uses the notation shown in Table I.

Parameter	Description
N	Number of disks in the storage system.
C	Number of cylinders on a disk.
SEC	Number of sectors on the disk.
SEC_c	Number of sectors on cylinder c.
spb	Number of sectors per block.
B	Logical request size (transfer block).
Q_i	Queueing time at disk i.
D_i	Seek distance on a disk i.
S_i	Seek time on a disk i.
R_i	Rotational latency on the same cylinder.
R_{MAX}	Full disk rotation time.
t	Block transfer time between the disk's buffer and its cylinder.
T	Bus transfer time of one block.
λ	Logical request arrival rate to the storage system.
p_i	Probability that disk i is used.
λ_i	Physical request arrival rate to disk i.
λ_{Rj}	Physical request arrival rate to the RAIDj area.
λ_{iRj}	Physical request arrival rate to a RAIDj area on disk i.
P_{raid_j}	$RAIDj$ area's proportion in the whole storage system space.
$Z_r(i)$	Read response time on disk i.
$Z_w(i)$	Write response time on disk i.
$\overline{Z_r}$	Mean response time for a read.
$\overline{Z_w}$	Mean response time for a write.
$\overline{Z}$	Mean response time for any request.
p_w	Probability that a request is a write.
p_r	Probability that a request is a read.
p_s	Probability of a sequential access.

TABLE I: Notation for the RAID model's parameters

A. Maximum of random variables

Suppose a task forks into a number of subtasks that are processed in parallel independently. The task's completion instant is that of the last subtask to complete processing, whereupon the subtasks combine (join) to re-form the original task. The Fork-Join time of the task, i.e. the time elapsed between the fork instant and the join instant, is therefore the maximum of the subtasks' processing times. In [6], the following result for the moments of Fork-Join times in a Markovian environment was derived:

Proposition 1: The kth moment $M_n(\boldsymbol{\alpha}, k)$ of the maximum of $n \geq 1$ independent, negative exponential random variables with parameters $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is defined by the recurrence

$$M_n(\boldsymbol{\alpha},k) = \frac{k}{\sum_{j=1}^{n}\alpha_j}M_n(\boldsymbol{\alpha},k-1)+\frac{\sum_{j=1}^{n}\alpha_j M_{n-1}(\boldsymbol{\alpha}_{\setminus j},k)}{\sum_{j=1}^{n}\alpha_j}$$

for $n \geq 1$ and $M_0(\boldsymbol{\epsilon},k) = 0$, for all $k \geq 1$, with $M_n(\boldsymbol{\alpha},0) = 1$ for all $n \geq 0$.
where $\boldsymbol{\alpha}_{\setminus j} = (\alpha_1,\ldots,\alpha_{j-1},\alpha_{j+1},\ldots,\alpha_m)$
In the special case that all the parameters of the exponential distributions are equal ($\forall j,\ \alpha_j = \alpha$) :

$$M_n(\boldsymbol{\alpha},1) = \frac{1}{\alpha}\sum_{m=1}^{n}\frac{1}{m} \quad and \quad M_n(\boldsymbol{\alpha},2) = \frac{1}{\alpha^2}\sum_{m=1}^{n}\frac{1}{m^2}$$

The logical request access time is defined as the maximum of all its physical request access times. We require the value of this quantity, assuming that the physical request access times are independent.

For generally distributed random variables, it is shown in [8] that the expected value of the maximum of n independent, non-negative random variables with means $\mathbf{m} = (m_1,\ldots,m_n)$, $\boldsymbol{\alpha}= (m_1^{-1},\ldots,m_n^{-1})$ and second moments $\mathbf{M} = (M_1,\ldots,M_n)$ is approximated by the function $I(n,\boldsymbol{\alpha},\mathbf{M})$ defined by the recurrence, for $k = 2,\ldots,n$, with $I(1,\alpha_1,M_1) = 1/\alpha_1$

$$I(k,\boldsymbol{\alpha},\mathbf{M}) = \frac{1}{k}\sum_{i=1}^{k} I(k-1,\boldsymbol{\alpha}_{\setminus i},\mathbf{M}_{\setminus i}) + \alpha_i M_i L_{k-1}(\boldsymbol{\alpha}_{\setminus i},\alpha_i)/2 \qquad (1)$$

where $L_{k-1}(\boldsymbol{\alpha}_{\setminus i},s)$ is the Laplace transform of the probability density function of the maximum of $k-1$ *exponential* random variables.

This result is exact if all the random variables are exponential. Notice that, when exact, all the summands give the same result. When approximate, the result is the average obtained by picking each of the k random variables in turn as the last in the sequence, and maximizing this and the maximum of the rest.

In the special case that all the parameters are equal $\forall i$, $\alpha_i = \alpha$ and $M_i = M$, proposition 1 gives the result

$$L_{k-1}(\boldsymbol{\alpha},\alpha) = 1/k$$

so that

$$I(k,\boldsymbol{\alpha},\mathbf{M}) = I(k-1,\boldsymbol{\alpha},\mathbf{M}) + \frac{M\alpha}{2k}$$

$$I(k,\boldsymbol{\alpha},\mathbf{M}) = 1/\alpha + (M\alpha/2)\sum_{i=2}^{k} 1/i$$

For each type of access (read or write), RAID variant and request size, the number of participating disks k is computed.

B. Mean response Time

Each disk is modelled by an M/G/1 queue of physical requests. It serves read/write requests and parity pre-read/update requests. Each physical request relates to one data block and its response time is composed of four terms, as formulated below. We use n overbars to indicate an nth moment.
1. **Queueing time** is calculated using the Pollaczek-Khinchin formulae[9], extended to handle multiple classes:

$$E[Q_i] = \frac{\sum_{j=1,5}\lambda_{iRj}\overline{\overline{X_{iRj}}}}{2(1-\rho_i)} \qquad (2)$$

where, refering to Table I,
- $X_i = S_i + R_i + t_i$ is the service time on disk i;
- t_i is the transfer time for one block from the track to the buffer of disk i;
- $\overline{\overline{X_{iRj}}}$ is the second moment of service time on $RAID_j$ area on disk i;
- $\rho_i = \lambda_i\overline{X_i}$ is the traffic intensity on disk i.

2. **Seek time**, depends on the distance D_i between the current position of the device's read/write head and the target position. In this paper, we use a more complex, but more accurate, formula [12] as follows :

$$S_i(D) = \begin{cases} 0 & \text{if } D_i = 0 \\ a\sqrt{D_i} + b(D_i-1) + c & \text{otherwise} \end{cases} \qquad (3)$$

where a, b, c are hardware-related constants :

$$a = (-10\times MinSeek + 15\times AvgSeek - 5\times MaxSeek)/(3\times\sqrt{C})$$

$$b = (7\times MinSeek - 15\times AvgSeek + 8\times MaxSeek)/(3\times C)$$

$$c = MinSeek$$

The hardware parameters *MinSeek, AvgSeek* and *MaxSeek* are respectively the seek time from one track to the adjacent one, the mean seek time and the seek time from the inner track to the outer. We assume that the incoming logical requests' addresses are independent random variables, uniformly distributed over the disk-address space. Since C is large, the distance D can be well approximated by a continuous random variable. Assuming that the number of sectors (and hence blocks) per track varies linearly with the cylinder number, the density function of D can be shown to be :

$$f_D(x) = A + Gx + Ex^3 \quad (0 \leq x \leq C-1) \qquad (4)$$

where we define the constants:

$$A = \frac{V(C-1)}{3\gamma^2}$$

$$G = -\frac{V+\beta^2(C-1)^2}{3\gamma^2}$$

$$
\begin{aligned}
E &= \frac{\beta^2}{3\gamma^2} \\
V &= 6\alpha^2 + 6\alpha\beta(C-1) + 2\beta^2(C-1)^2 \\
\gamma &= \alpha(C-1) + \beta(C-1)^2/2 \\
\alpha &= SEC_{C-1}/spb, \\
&\quad \text{the number of blocks on an innermost track} \\
\beta &= SEC_0/spb - SEC_{C-1}/spb, \\
&\quad \text{the difference between the number of} \\
&\quad \text{blocks on outermost and innermost tracks.}
\end{aligned}
$$

For simplicity, we have assumed that all disks have the same hardware parameters, including a, b, c, C. However, it would be easy to extend our model to heterogeneous devices.

3. **Rotational latency** is a random variable with uniform distribution on $[0, R_{MAX}]$, with density function :

$$f_R(x) = \frac{1}{R_{MAX}} \qquad 0 \le x \le R_{MAX} \qquad (5)$$

4. **Single block transfer time** is composed of a bus transfer time T and a cylinder/device_buffer transfer time t. Assuming negligible contention, T depends only on the bus bandwidth. Previously, t was considered fixed, but this assumption is no longer valid on multi-zoned disks. In fact, t depends on the position of the accessed cylinder on the disk. There are more sectors on outer zones leading to a reduced block transfer time. Assuming track-alignment, the transfer time of a block on a cylinder c is calculated as:

$$\frac{spb * R_{max}}{SEC_c}$$

C. Moments

The first three moments of the four parameters above are needed in the analytical model and calculated as :

1. **Queueing time :** Multi-zoning has no effect on the queueing time formula itself – it just affects the server utilization, service time variance, and hence the queueing time values given by that formula. Thus the moments' expressions are unchanged from [8].

2. **Seek time :** This is the parameter most affected by multi-zoning. Its new moments are given by :

$$
\begin{aligned}
\overline{S} &= (c-b) + aM_{1/2} + bM_1 \\
\overline{\overline{S}} &= (c-b)^2 + 2a(c-b)M_{1/2} \\
&\quad +[a^2 + 2b(c-b)]M_1 + 2abM_{3/2} + b^2M_2 \\
\overline{\overline{\overline{S}}} &= (c-b)^3 + a(c-b)^2M_{1/2} \\
&\quad +3(c-b)[a^2 + b(c-b)]M_1 \\
&\quad +[a^3 + 6ab(c-b)]M_{3/2} \\
&\quad +3[a^2b + b^2(c-b)]M_2 + 3ab^2M_{5/2} + b^3M_3
\end{aligned}
$$

where the nth moment of D is, for $n \ge 0$ (not necessarily integer):

$$M_n = (C-1)^{n+1}\left[\frac{A}{n+1} + \frac{G(C-1)}{n+2} + \frac{E(C-1)^3}{n+4}\right]$$

3. **Rotational latency :** Multi zoning has no effect on the rotational latency; thus the moments' expressions are unchanged from [8].

4. **Cylinder/device_buffer transfer time :** It is not constant and its first three moments can be shown to be :

$$\overline{t} = \frac{spb * R_{max} * (C-1)}{SEC}$$

$$\overline{\overline{t}} = \frac{spb^2 * R_{max}^2}{SEC} \times \Sigma_{j=0}^{C-1}\frac{1}{SEC_j}$$

$$\overline{\overline{\overline{t}}} = \frac{spb^3 * R_{max}^3}{SEC} \times \Sigma_{j=0}^{C-1}\frac{1}{SEC_j^2}$$

From the above, we calculate the moments of the positioning time Y and hence of the service time X:

$$\overline{X} = \overline{Y} + \overline{t}$$

$$\overline{\overline{X}} = \overline{\overline{Y}} + \overline{\overline{t}} + 2\overline{Y}\,\overline{t}$$

$$\overline{\overline{\overline{X}}} = \overline{\overline{\overline{Y}}} + \overline{\overline{\overline{t}}} + 3\overline{\overline{Y}}\,\overline{t} + 3\overline{Y}\,\overline{\overline{t}}$$

IV. Simulation

To evaluate our multi-RAID model for zoned disks, we adapted the simulator of [8] to handle the new disk geometry and access time functions. The simulator is written in C and is composed of three main parts: a logical request generator, a logical to physical mapping core and a simulation engine. The hardware parameters are obtained from a library, which is separated from the execution routines for flexibility and scalability purposes. Two modules were especially affected by the modification : the address mapping and the access time calculation modules. Experiments were carried out using different combinations of both workload and architecture configuration parameters.

The RAID array modelled is composed of 16 disks connected to an ultra wide SCSI bus. The characteristics of the zoned disk used in the simulations are summarized in table II. Additional details on this device can be consulted in [4].

Parameter	value
Formatted capacity	36,74 GB
Sectors/device (SEC)	$18,37 \times 10^7$
Rotation	10000 rpm
Cylinders (C_{max})	29950
Min Seek	0,4 - 0,6 ms
Avg Seek	4,5 - 5 ms
Max Seek	11 - 12 ms
Data Heads	4
Zones number	18

TABLE II: Disk Characteristics - Fujitsu-MAN3367

Simulations were run for a warm-up period of 300000 logical request arrivals, then for a further 700000 arrivals, during which the measurements were gathered.

V. Results and discussion

To compare the adapted Multi-RAID model (with zoned disks) against the simulation, we plotted their respective mean queueing and response times. We validated our model according to four main parameters :
1. **Request size :** Figures 2 and 3 show the effect of the request size, in terms of the number of blocks per logical request (B), on the mean queueing and response times respectively. The RAID01 variant was chosen for this experiment in view of its popularity due to a lower cost of disks. The access request environment was exclusive read to isolate the basic disk access behaviour from any complex write execution scheme. Good agreement is apparent on both figures.

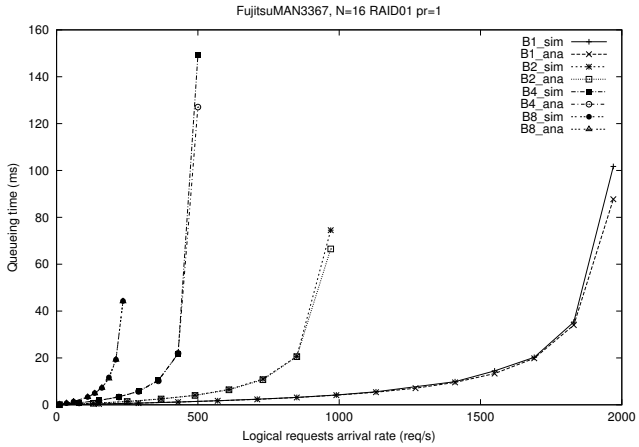


Fig. 2. Queueing time (RAID01,pr=1)

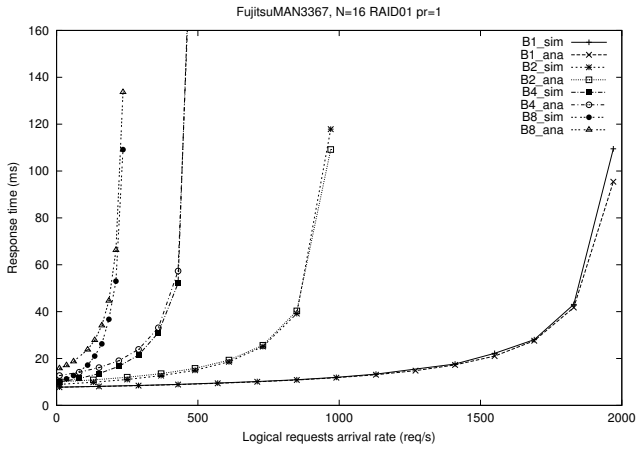


Fig. 3. Response time (RAID01,pr=1)

2. **Request type :** In figures 4 and 5, we see the effect of the request type (read/write) on the queueing and response times respectively, in a RAID01 environment with small requests (B=1). We focused on the small request size (4KB blocks) because it is the size of 96% of requests in an OLTP workload [10], typically associated with large storage systems. In addition to the good agreement, we notice the effect of the double disk access generated by every write request on the mean response time.

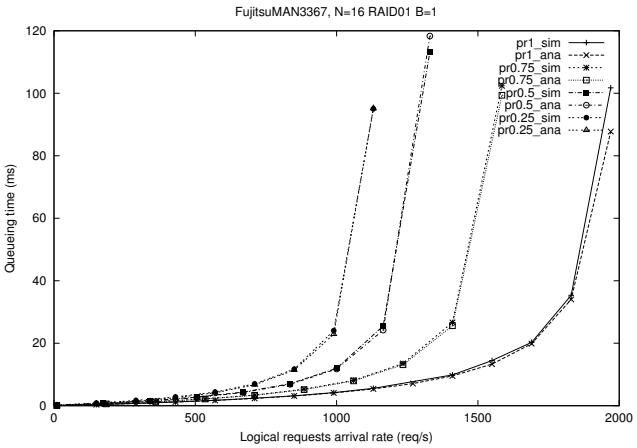


Fig. 4. Queueing time (RAID01,B=1)

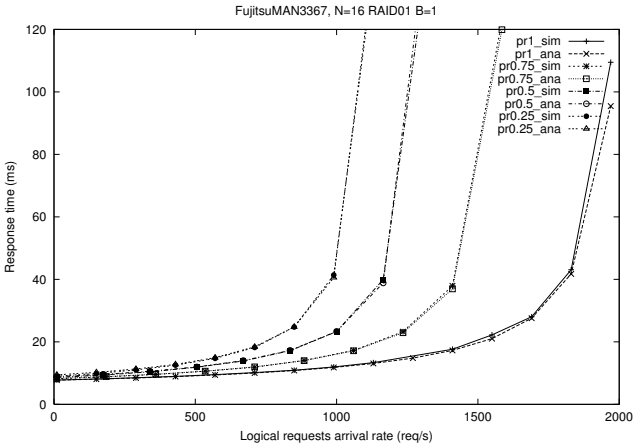


Fig. 5. Response time (RAID01,B=1)

3. **RAID variant :** Figures 6 and 7 show the effect of the RAID variant on the queueing and response times respectively. In fact, these figures show the RAID5 performance, considering some combinations of request type, to be superior to (i.e. with lower queueing and response times than) that of RAID01 on figures 4 and 5. Notice again the good agreement with simulation.

The original model [6], [25] was motivated by the need for an analytical model of a Multi-RAID storage system[1], when controlling asynchronous disks with uniformly distributed data. Figures 8 and 9 respectively show the mean queueing and response times for a mixed workload (75% reads) and a Multi-RAID (mixed RAID) system with two different mixture ratios : a RAID01 oriented system (75% of RAID01) and a RAID5 oriented system (75% of RAID5). Also shown on these figures are the corresponding exclusive RAID01 and exclusive RAID5 results as bounds. The agreement found confirms that the model remains

[1]Where many RAID schemes coexist on the same storage space. Our study is limited to the RAID01 and RAID5, the most commonly used ones.

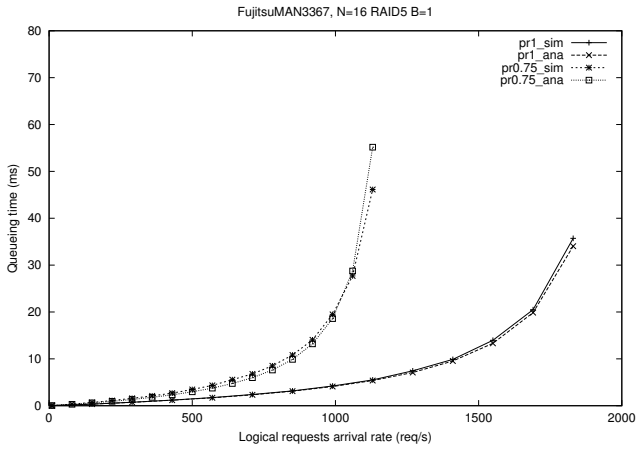


Fig. 6. Queueing time (RAID5,B=1)

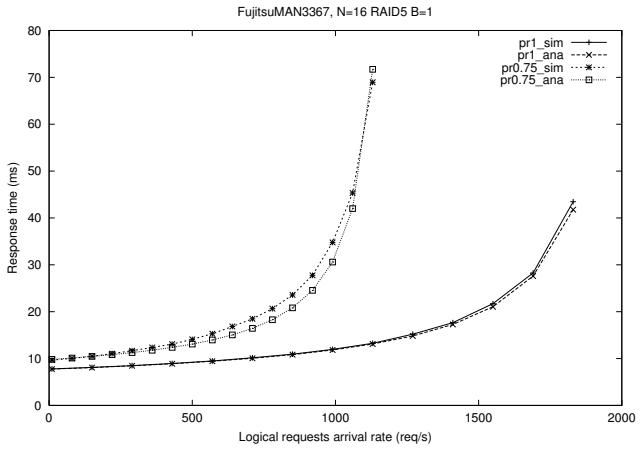


Fig. 7. Response time (RAID5,B=1)

valid and accurate for Multi-RAID storage systems when extended to modern zoned disks.

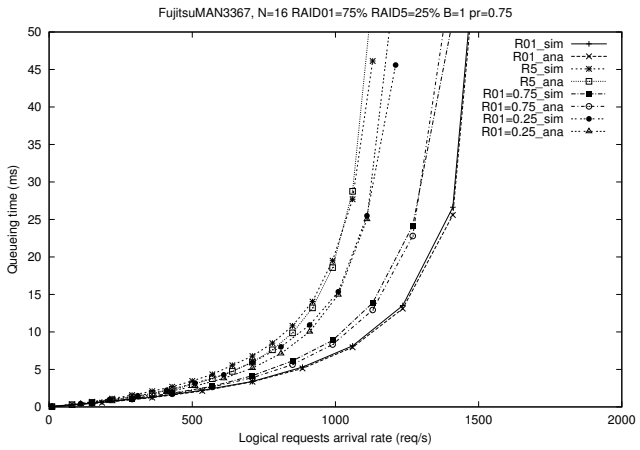


Fig. 8. Queueing time (R01/R5)

In general, we see good agreement between the analytical results and the simulation, as well as the decreasing penalizing effect of the small RAID5 writes on mean response time, as its usage percentage decreases in the mixed RAID system.

4. **Quantitative model accuracy:** We estimated how accurate are the various parts of our model for multi-zoned disks by analysing each component of the

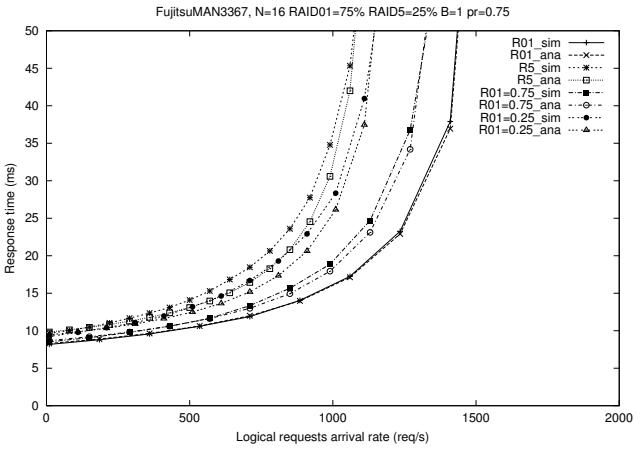


Fig. 9. Response time (R01/R5)

access time separately. We can see, in tables III and IV, the excellent precision of the first two moments of the seek distance (D), the first three moments of the seek time (S) and the rotational latency (R), respectively.

Moment	Model	Simulation	% err
1	9866.27	9795.23	0.72%
2	146 577 339.47	144 211 521.78	1.6%

TABLE III: Moment comparison:seek distance (D)

	Moment	Model	Simulation	% err
S	1	4.708	4.683	0.53 %
	2	28.54	28.24	1.06 %
	3	200.35	198.1	1.13 %
R	1	3.00	2.993	0.23 %
	2	12.0	11.949	0.42 %
	3	54.0	53.681	0.59 %

TABLE IV: Moments comparison:seek time (S) and rotational latency (R)

VI. Conclusion

We developed an intricate analytical model in [6], [25] for a multi-RAID storage system, based on queueing theory and taking into account the effect of synchronized Fork-Join operations. In this paper, we took a step further by adapting that model to modern, zoned disk technology. This is of great interest because the model is completely free of simulation and estimation-approximations – although obviously not of its approximating assumptions, which have been carefully checked [7]. All the constituent moments have been calculated directly, giving additional accuracy. We validated our new model against simulation results, considering different combinations of request sizes, request types and RAID variants in their mixtures. The excellent agreement we obtained suggests our model is a suitable tool to evaluate, in a short time compared to simulation, the performance of any configuration of a multi-RAID storage system, without requiring any approximated input parameters.

In the near future, we plan to compare our model with

those developed by [15], [22], to estimate the accuracy obtained analytically as opposed to using interpolation and other approximations. We intend also to handle heterogeneous disks in the same storage system to increase the flexibility and scalability of our model.

References

[1] S. Chen and D. Towsley. A performance evaluation of raid architectures. *IEEE Transactions on Computers*, 45(10), October 1996.

[2] G. Gibson D. A. Patterson and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of SIGMOD Conference*, June 1988.

[3] J. Xu E. Varki, A. Merchant and X. Qiu. An integrated performance model of disk arrays. In *Proceedings of the International Symposium onModelling, Analysis and Simulationof Computer and Telecommunications Systems (MASCOTS)*, 2003.

[4] Fujitsu. *Disk Drives. Products/Maintenance Manual*. Fijitsu, 2001.

[5] C. Han and A. Thomasian. Performance of two disk failure tolerant disk arrays. In *International Symposium on performance Evaluation of computer and telecommunication systems*, 2003.

[6] P.G. Harrison and S. Zertal. Queueing models with maxima of service times. In *Proceedings of TOOLS Conference*, 2003.

[7] P.G. Harrison and S. Zertal. Calibration of a queueing model of raid systems. In *Proceeding of 2004 International Workshop on Practical Applications of Stochastic Modelling*, 2004.

[8] P.G. Harrison and S. Zertal. Queueing models of RAID systems with maxima of waiting times. *Performance Evaluation*, 2006. To appear.

[9] Ng Chee Hock. *Queueing Modelling Fundamentals*. John Wiley Publisher, 1996.

[10] P. Biswas K. K. Ramakrishnan and R. Karedla. Analysis of file I/O traces in commercial computing environments. In *Proc. Joint conf. on Measurement and Modelling of Computer Systems SIGMETRICS/PERFORMANCE*, June 1992.

[11] M. Y. Kim and A.N. Tantawi. Asynchronous disk interleaving: approximating access delays. *IEEE Transactions on Computers*, 40(7), 1991.

[12] E.K. Lee. *Perfomance modelling and analysis of disk arrays*. ph.D. thesis, University of California, Berkeley, USA, 1993.

[13] A. Merchant and P.S. Yu. An analytical model or reconstruction time in mirrored disks. *Performance evaluation*, 20, May 1994.

[14] A. Merchant and P.S. Yu. Analytic modeling of clustered raid with mapping based on nearly random permutation. *IEEE Transactions on Computers*, 45(3), March 1996.

[15] R. Nelson and A. N. Tantawi. Approximate analysis of fork-join synchronisation in parallel queues. In *IEEE Trans. Computers*, volume 37, June 1998.

[16] S. Christodoulakis P. Triantafillou and C. A. Georgiadis. A Comprehensive Analytical Performance Model for Disk Devices Under Random Workloads. *IEEE Transactions on Knowledge and data Engineering*, 14(1), 2002.

[17] Sangsoo Park and Heonshik Shin. *Rigourous Modeling of Disk Performance for Real-Time Applications*, volume 2986. Springer Berlin, 2004.

[18] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3), March 1994.

[19] A. Thomasian and A. N. Tantawi. Approximative solutions for M/G/1 Fork/Join synchronisation. In *Proceedings of the 1994 Winter Simulation Conference*, 1994.

[20] E. Varki. Mean Value Technique for Closed Fork-Join Networks. In *Proceedings of ACM SIGMETRICS conf. on Measurement and Modeling of Computer Systems*, May 1999.

[21] E. Varki and L. W. Dowdy. Analysis of Balanced Fork-Join Queueing Networks. In *Proceedings of ACM SIGMETRICS conf. on Measurement and Modeling of Computer Systems*, May 1996.

[22] S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork/join queues. In *Performance Evaluation Journal*, volume 20, 1994.

[23] T.M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *USENIX Ann. Technical Conference*, 2002.

[24] S. Zertal. *Dynamic redundancy mechanisms for storage customisation on multi disks storage systems*. ph.D. thesis, University of Versailles, France, January 2000.

[25] S. Zertal and P.G. Harrison. Multi-level raid storage system modelling. In *Proceedings of 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (Spects)*, 2003.

BSP/CGM Algorithms for the Transitive Closure Problem

Edson Norberto Cáceres
Cristiano Costa Argemom Vieira
Federal University of Mato Grosso do Sul
Dept. de Computação e Estatística
Campo Grande - MS, Brazil
E-mail:{edson,ccav}@dct.ufms.br

Abstract— **We present new implementations of BSP/CGM algorithms for the Transitive Closure Problem. Our strategies deal with size of the message and communication rounds, problems that cause inefficiency in the implementations of the transitive closure algorithms. The algorithms were implemented using LAM/MPI in two Beowulfs. The implementation results show the efficiency and scalability of the presented algorithms, improve the previous results and compare favorably with other parallel implementations. Besides we show that the presented algorithms can be used to solve real problems.**

I. Introduction

The search for efficient algorithms to compute the *transitive closure* of a directed graph (*digraph*) has been around for many years. It was first considered in 1959 by in the paper [Roy, 1959]. The best sequential algorithms that solve this problem have $O(mn)$ time complexity, where m and n are the number of edges and vertices of the digraph [Habib and Rampom, 1993] and [Simon, 1988]. There are other algorithms that are based on matrix multiplication that can be used for dense digraphs.

Using an approach based on the adjacency matrix of the digraph [Warshall, 1962] presented an $O(n^3)$ algorithm. This algorithm can be implemented using an adjacency bit matrix [Baase, 1978] with $O(\frac{n^3}{\alpha})$ time complexity, where α is the size of bits that can be stored in a primitive data.

Another approach to compute the transitive closure is using digraph traversal. Using *breadth first search* BFS (or *depth first search* DFS) we can find all the vertices that are reachable from a given vertex v. Applying this search to each vertex of the digraph we can compute the transitive closure. A BFS (or DFS) can be computed in $O(n+m)$[Cormen et al., 2001] time complexity. Then the transitive closure can be computed in $O(n\ (n+m))$ time complexity.

Parallel algorithms for this problem have been proposed to the PRAM model [JáJá, 1992, Karp and Ramachandran, 1990] BSP/GCM model [Tiskin, 2001, Cáceres et al., 2002, Alves et al., 2003, Cáceres and Vieira, 2004] and other architectures [Pagourtzis et al., 2001, 2002, Grama et al., 2003]. Tiskin observes that "is not clear yet whether the presented algorithm [Tiskin, 2001] is practical".

Using the BSP/CGM model, [Cáceres et al., 2002] presented an algorithm to compute the transitive closure of acyclic digraph using $\log p+1$ communication rounds where p is the number of processors and uses $O(mn/p)$ local computation. This algorithm assumes that the input is an acyclic digraph and relies on the so-called linear extension labeling of the input digraph vertices.

Using the ideas of this algorithm and the sequential algorithm of Warshall, [Alves et al., 2003] implemented a transitive closure algorithm with the MPI library in a Beowulf with 64 nodes with good speed-up. This implementation do not compute the linear extension of the digraph, and it can spend p communication rounds in the worst case, and uses $O(n^3/p)$ local computation. They describe the worst case where p communication rounds are necessary. With randomly generated digraphs, in their experiments, the algorithm always found the transitive closure with $O(\log p)$ communication rounds. Besides the fact of the good speed up, in this algorithm, the processors exchange huge amount of data ($O(n^2/p)$ in the worst case) in each communication round. Nevertheless, it is very fast and it obtained better execution times than the previous ones ([Pagourtzis et al., 2001, 2002]).

Combining the ideas of [Alves et al., 2003] and [Baase, 1978]. [Cáceres and Vieira, 2004] implemented a version of transitive closure algorithm with $\frac{p}{\alpha}$ communication rounds and $O(\frac{n^3}{\alpha p})$ local computation, where where α is the size of bits that can be stored in a primitive data. Since in this implementation, as p grows, the local computation is very fast, the total time of the algorithm is bounded by the communication time.

In this paper we deal with the main problems of the previous BSP/CGM algorithms for the transitive closure problem: the size of the messages, the local computation time and the number of communication rounds. We present four new implementations. The first and second deal with the size of the messages and using data compression we can decrease the amount of data exchanged in each communication round to $O(n/p)$. The third uses Bit matrix to improve the local computation time. The fourth increases the amount of memory used by each processor and can solve the transitive closure with a constant number of communication rounds. This algorithm uses a graph traversal strategy to compute the transitive closure of the input digraph.

II. Notation and Computational Model

Let $D=(V,E)$ be an acyclic digraph, $n=|V|$ and $m=|E|$ the number of vertices and edges of D, respectively. The vertices $v_i \in V$ are labeled with $1 \leq i \leq n$. We use both digraphs representations: adjacency list and adjacency matrix.

Partially supported by CNPq, FUNDECT-MS and CAPES

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

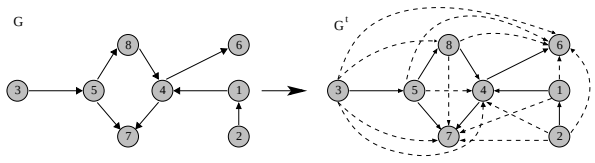


Fig. 1. Transitive Closure D^t of digraph D.

We denote a path between vertices $v_i, v_j \in V$ as follows: *(i)* $v_i \rightarrow v_j$, a path consisting of only one edge starting at vertex v_i and ending at vertex v_j.
(ii) $v_i \rightsquigarrow^k v_j$, a path starting at vertex v_i and ending at vertex v_j with k intermediate vertices ($k+1$ edges).
(iii) $v_i \rightsquigarrow^{\cdots k} v_j$, a path starting at vertex v_i and ending at vertex v_j with $v_1, v_2, ..., v_k$ as intermediate vertices.

When necessary, we denote an edge $e \in E$ as $e_{ij} = v_i \rightarrow v_j$. The edge e_{ij} is a leaving edge with respect to v_i and an incoming edges with respect to v_j.

Let $\Phi_q = \{q\frac{n}{p} + 1, .t.., (q+1)\frac{n}{p}\}$ be the set of the $\frac{n}{p}$ consecutive vertices that each processor receives in order to compute the transitive closure.

The transitive closure D^t of a digraph $D = (V, E)$ is obtained from D by adding an edge (v_i, v_j) if there is a path $v_i \rightsquigarrow^k v_j$, $0 < k \leq n$, in D. Figure 1 represents a digraph and its respective transitive closure.

In this paper, we use the BSP/CGM Model (Bulk Synchronous Parallel/Coarse Grained Multicomputer) [Valiant, 1990, Dehne, 1999]. Let N be the input size of the problem. A BSP/CGM computer consists of a set of p processors each with local memory and each processor is connected by a router that can send/receive messages in a point-to-point fashion. A BSP/CGM algorithm consists of alternating local computation and global communication rounds separated by a barrier synchronization.

In a computing round, we usually use the best sequential algorithm in each processor to process its data locally. We require that all information sent from a given processor to another processor in one communication round be packed into one long message, thereby minimizing the message overhead. In the BSP/CGM model, the communication cost is modeled by the number of communication rounds. Each processor can send/receive at most $O(N/p)$ data in each communication round.

Finding an efficient algorithm on the BSP/CGM model is equivalent to minimizing the number of communication rounds as well as the total local computation time.

III. The Parallel Algorithms

Now we present the three new BSP/CGM parallel algorithms for the transitive closure problem. First we deal with the size of the messages that are exchanged between the processor. Then we decrease the local computation time and finally we design an algorithm that does not use communication rounds.

A. Dealing with Message Size - I

Analyzing the previous transitive closure algorithms we observed that the implementation presented by [Alves et al., 2003] and [Cáceres and Vieira, 2004] the communication cost is very expressive. In most cases, the messages have $O(n^2/p)$ size. One of the causes is the fact that the "new" edges are exchanged by the processors during the communication rounds without any further consideration.

We introduce a very simple technique that shrinks expressively the size of the messages that are exchanged by the processors. We define a *super-vertex* the set of all reachable vertices from v_i, i.e., $S_{v_i} = \{v : v_i \rightarrow v\}$. When a vertex v_i does not have any leaving edge, we denote $S_{v_i} = \varnothing$.

Let $V = \{v_i : 1 \leq i \leq n\}$ the set of vertices of D. We label the vertices v_i with labels $(v_i) = 2^{i-1}$ and we define $N_{v_i} = \sum_{\{v \in S_{v_i}\}} (v)$.

The Algorithm 1 shows the N_{v_i} computational steps for a S_{v_i} input and the Algorithm 2 shows the reverse (N_{vi} to S_{v_i}).

Algorithm 1 BuildN_{v_i}

Input: S_{v_i}
Output: $N_{v_i} = \sum S_{v_i}$.
1: $N_{v_i} \leftarrow 0$
2: **for all** $v_x \in S_{v_i}$ **do**
3: label(v_x) $\leftarrow 2^{x-1}$
4: **end for**
5: **for all** $v_x \in S_{v_i}$ **do**
6: $N_{v_i} \leftarrow N_{v_i}$ + label(v_x)
7: **end for**

It is easy to see that this labeling technique (Algorithm 1) guarantees a unique representation for each vertex. For example, if $S_{v_i} = \{1, 3, 4\}$, then the labels are $(v_1) = 1$, $(v_3) = 4$, $(v_4) = 8$ and $N_{v_i} = 13$. There is no other combination of the labels such that the sum is 13.

Algorithm 2 RestoreS_{v_i}

Input: N_{v_i}.
Output: S_{v_i}.
1: $S_{v_i} \leftarrow O$
2: $x \leftarrow 1$
3: **while** $x < N_{v_i}$ **do**
4: $x \leftarrow 2x$
5: **end while**
6: $x \leftarrow \frac{x}{2}$
7: **while** $N_{v_i} > 0$ **do**
8: **if** $N_{v_i} - x \geq 0$ **then**
9: $N_{v_i} \leftarrow N_{v_i} - x$
10: $S_{v_i} \leftarrow S_{v_i} \cup \{v_x\}$
11: **end if**
12: $x \leftarrow \frac{x}{2}$
13: **end while**

The restore procedure (Algorithm 2) computes the original values of the vertices, the set S_{v_i}. This is done by repeatedly subtracting (v_x) from N_{v_i}, starting with the greatest value of (v_x) to the smallest. The loop starting at the line **3** computes the greatest label that was stored in N_{v_i}.

In the [Alves et al., 2003] algorithm, each processor p_i receives the sub-matrices $D[(j-1)\frac{n}{p} + 1..j\frac{n}{p}][1..n]$ and $D[1..n][(j-1)\frac{n}{p} + 1..j\frac{n}{p}]$ and compute the local transitive closure. Before each processor p_i sends the computed transitivity edges to processor p_j, we apply algorithm 1, to

the message, creating a super-vertex for each line/column of the sub-matrix, and then send the compressed message to processor p_j. After processor p_j receives a message with super-vertices from processor p_i it applies the algorithm 2 restoring the original values of the vertices.

We describe the results of the implementation of this technique in Section IV. The size of the messages shrunk significantly decreasing the time of each communication round.

B. Dealing with Message Size - II

Now we introduce another simple technique that shrinks expressively the size of the messages that are exchanged by the processors.

For each new edge w_{ij}, $0 \leq i, j, < n$, the values of i and j (row and column) must be packed and during the communication rounds processor p_k sends the new edges that are in the l-th vertical strip and l-th horizontal strip to the processor p_l. This can be done in two ways:
i) Each edge can be represented as a value a where $a = i * n + j$.
ii) We represent each strip with a bit matrix where each edge can be stored as a bit.

Let Q^k be the number of new edges that will be send by processor p_k. Before each communication round we verify if we will send only the new edges or the whole strips (vertical and horizontal).

The Algorithm 3 implements the above ideas.

Algorithm 3 Parallel-Warshall-II

Input: D given as an $n \times n$ matrix W; p the number of processors; and each processor p_z ($0 \leq z \leq p-1$) stores the submatrix $W[(z\frac{n}{p}+1..(z+1)\frac{n}{p}][1..n]$ and $W[1..n][z\frac{n}{p}+1..(z+1)\frac{n}{p}]$
Output: D^t represented as W^t distributed by the p processors.

```
1:  repeat
2:    for k ← l(n/p) + 1 to (l + 1)(n/p) − 1 do
3:      for i ← 1 to n do
4:        for j ← 1 to n do
5:          if w_ik = 1 and w_kj = 1 then
6:            w_ij ← 1
7:          end if
8:        end for
9:      end for
10:   end for
11:   if Q^k > n^2/c then
12:     Send/Receive only the edges belongs another processor
13:   else
14:     Send/Receive the full strips another processor
15:   end if
16: until no new edges are computed
17: return D^t
```

Theorem 1: The Algorithm 3 uses $O(\frac{n^3}{p})$ local computation time with $O(p)$ communication rounds and needs $O(\frac{n^2}{p})$ local space.

We describe the results of the implementation of this technique in Section IV. The size of the messages shrunk significantly decreasing the time of each communication round.

C. Improving the Local Computation Time by Using the Bit Matrix

Representing the digraph with a bits adjacency matrix, [Cáceres and Vieira, 2004] implemented the transitive closure algorithm with expressive results. The implementation decreased significantly the local computation. The only drawback is that they use $O(n^2)$ local memory. Using $O(n^2)$ local memory we can sequentially compute the linear extension of the digraph in one processor and assure that with $\lg p+1$ communication rounds the algorithm computes the transitive closure. Besides the linear extension computation we can aggregate the ideas of Algorithm 3 and implement a new version of the [Cáceres and Vieira, 2004] algorithm.

The Algorithm 4 describes the above ideas.

Algorithm 4 PTC-Bit

Input: D given as an $n \times n$ bit matrix W; p the number of processors; and each processor p_z ($0 \leq z \leq p-1$) stores a copy of W.
Output: D^t represented as W^t distributed by the p processors.

```
1:  if z=0 then
2:    Compute L and Φ_{0,...,p−1}
3:  end if
4:  repeat
5:    for all k ∈ Φ_z in order do
6:      for i ← 1 to n do
7:        if w_ik = 1 then
8:          for j = 1 to n/α do
9:            w_ij ← w_ij ∨ w_kj
10:         end for
11:       end if
12:     end for
13:   end for
14:   if Q^k > n^2/c then
15:     Send/Receive only the edges belongs another processor
16:   else
17:     Send/Receive the full strips another processor
18:   end if
19: until no new edges are computed
20: return D^t
```

Because the vertex distribution to obey a linear extension of the digraph, $O(\log p)$ communication rounds are enough [Cáceres et al., 2002].

Theorem 2: The Algorithm 4 uses $O(\frac{n^3}{p\alpha})$ local computation time with $O(\log p)$ communication rounds and needs $O(n^2)$ local space.

D. Avoiding Communication Rounds

Since the amount of communication spent by the previous algorithms is very large, and limits the performance, we tried to trade communication with local space. Using the Warshall approach, seems that no improvement can be obtained with more local space. Actually, in most cases, increasing the local space does not guarantee any improvement to the parallel algorithm.

Using the digraph traversal (BFS or DFS) approach to compute the transitive closure with $O(n^2/p)$ local memory seems difficult, since parallel algorithms for these searches are not efficient. But we devise that if each processor stores the whole adjacency list of the digraph, each processor can apply sequential BFS or DFS to its n/p vertices. Then, processor p_l computes the transitive closure of the vertices in Φ_l, without any communication with other processors.

Let Γ^i be the spanning tree obtained by the breadth first search in the digraph D starting at the vertex v_i, the Algorithm 5 presents the steps for computing the transitive closure D^t.

Algorithm 5 TCP-BFS

Input: adjacency list Q of D
Output: D_l^t
1: **for all** $v_i \in \Phi_l$ **do**
2: $Q_i^t \leftarrow NULL$
3: Compute Γ^i.
4: Build the edge $q_{i,j}^t = \{(v_i, v_j) : j \in \Gamma^i\}$
5: **end for**

At the end of the algorithm, the transitive closure will be distributed by the processors.

Theorem 3: The BSP/CGM Algorithm 5 computes the transitive closure D^t of the input digraph D with $O(\frac{n}{p}(n + m))$ local computation, $O(n/p+m)$ local space and without any communication rounds.

Besides the fact that different processors can compute the same transitivity edge (we do not have communication rounds), the implementation of the algorithm showed an expressive performance when compared with the algorithms that use Warshall's approach.

IV. The Implementation Results

The digraphs used in the tests were generated randomly with probabilities varying from 15% to 20% of existing an edge between any two graph vertices.

TABLE I: **I**-Execution times of [1], **II**-Execution times of the Super-Vertex Algorithm and **III**-Execution times of the BFS Algorithm. ⋆(Alg.A with 1920 x 1920)

	512 x 512			1024 x 1024			2048 x 2048⋆		
Procs.	**I**	**II**	**III**	**I**	**II**	**III**	**I**	**II**	**III**
1	25.4	10.6	3.71	143	84.7	30	1614	679	232
2	9.3	5.3	1.87	123	42.7	15	603	341	116
4	8.3	2.8	0.93	84.4	21.7	7.6	257	171	58.2
8	3.9	1.5	0.47	36.4	11.3	3.8	123	87.7	29.1
16	2.6	1.3	0.24	18.0	6.0	1.9	69	45.6	14.5
32	1.9	0.9	0.12	15.6	6.5	0.9	68	24.4	7.2
64	3.3	0.7	0.06	12.1	7.5	0.4	49	24.4	3.6

We implemented the algorithms on two Beowulfs (clusters).

The first one with 64 nodes consisting of low cost microcomputers with 256MB RAM, 256MB swap memory, CPU Intel Pentium III 448.956 MHz, 512KB cache, in addition to two access nodes consisting of two microcomputers each with 512MB RAM, 512MB swap memory, CPU Pentium 4 2.00 GHz, and 512KB cache. The cluster is divided into two blocks of 32 nodes each. The nodes of each block are connected through a 100 Mb fast-Ethernet switch. Each of the access nodes is connected to the switch that connects the block of nodes and the two switches are connected. This cluster is the same used in [Alves et al., 2003].

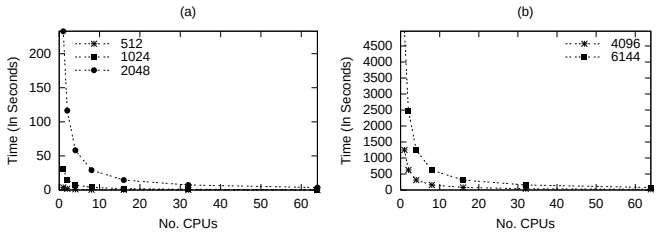


Fig. 2. Execution times of the BFS Algorithm

The second cluster is composed by 12 nodes consisting of 6 CPU Intel Pentium IV of 1.7Ghz and 6 CPU AMD Athlon de 1.6GHz. The nodes are connected by a 1Gb fast-Ethernet switch.

Our code is written in standard ANSI C++ using the LAM-MPI library. The execution times are expressed in seconds.

The Table IV presents a comparison between the implementation presented by [Alves et al., 2003] and our implementations.

The column II describe the times that uses Algorithm 1 an Algorithm 2 (super-vertex strategy). Its implementation got better execution times than the results presented by [Alves et al., 2003]. With the super-vertex strategy, actually we do a compression on the messages that are exchanged by the processors. In our tests we represented N_v with a data type with $\alpha = 32$ bits. The column III uses Algorithm 5, the breadth first search approach. The implementations were executed in digraphs with 512, 1.024, 2.048 and 4096 vertices and 50.000, 210.000, 800.000 and 3.300.000 edges, respectively.

The Table IV describes the results of [Alves et al., 2003] (I), and our implementation of Algorithm 3 (II-III). The Table IV describe the results of [Cáceres and Vieira, 2004] (I) and our implementation (II-III). Our results are much faster than the previous implementations.

The implementations were executed in digraphs with 1.024, 2.048 and 4.096 vertices and 210.000, 800.000 and 3.300.000 edges, respectively.

The Figure 3 shows the curves of Table IV.

The Figure 4 shows the curves of Table IV.

The Figure 5 shows the comparison of the communication times with/without using bit arrays sending/receiving messages in the two clusters.

V. Conclusions

We presented four new implementations of BSP/CGM parallel algorithm for the transitive closure. The first and second deal with the size of the messages that are exchanged

TABLE II: **I**-Execution times of [1], **II**-Execution times of the PTC-Edge Insertions Algorithm on the first cluster and **III**-Execution times of the PTC-Edge Insertions Algorithm on the second cluster. *(I - with 1920 x 1920)

	1024 x 1024			2048 x 2048			4096 x 4096	
Procs.	**I**	**II**	**III**	**I**	**II**	**III**	**II**	**III**
1	143	86	23	1614	688	190	5300	1521
2	123	44	12	603	348	100	2780	788
4	84	22	6	257	176	50	1427	407
8	36	12	3	123	91	27	723	206
16	18	6	-	69	48	-	377	-
32	15	5	-	68	26	-	205	-

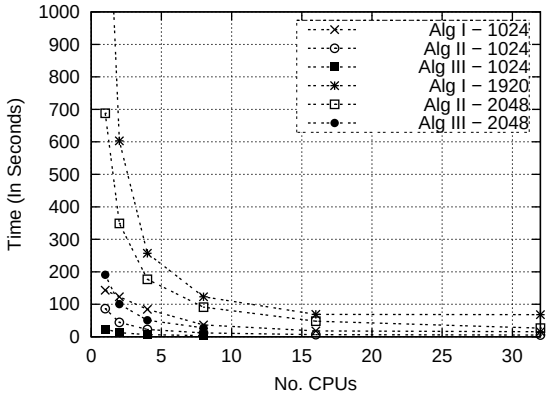


Fig. 3. Curves of Table IV

between processor. We introduced the techniques of of super-vertex and bit arrays that shrink the messages size. With these simple techniques we obtained a better performance implementing a BSP/CGM transitive closure algorithm. Then we notice that in the Cáceres and Vieira one processor could compute the linear extension of the input digraph and apply the same idea of the first implementation to decrease the size of the messages. Finally we implemented a BSP/CGM algorithm that uses a BFS search to compute the transitive closure of a digraph instead of the Warshall approach.

We implemented the algorithms in two Beowulfs with 64 and 12 nodes, respectively, using LAM-MPI. Since the BFS Algorithm does not use communication rounds it has the best performance between the proposed algorithms. The implementation results show the efficiency and scalability of the presented algorithms, improve the previous results and compare favorably with other parallel implementations.

TABLE III: **I**-Execution times of [4], **II**-Execution times of the PTC-BIT Algorithm on the first cluster. and **III**-Execution times of the PTC-BIT Algorithm on the second cluster

	1024 x 1024			2048 x 2048			4096 x 4096	
Procs.	**I**	**II**	**III**	**I**	**II**	**III**	**II**	**III**
1	7.7	7.6	2.1	63.3	61.6	17.5	505.3	134.6
2	5.2	3.9	1.2	36.4	31.3	9.9	254.4	77.8
4	4.0	1.9	0.7	23.7	15.8	5.1	127.9	39.5
8	3.4	1.0	0.3	17.4	8.1	2.6	64.7	20.5
16	3.7	0.5	-	18.3	4.4	-	33.2	-
32	3.5	0.4	-	15.2	2.5	-	17.9	-

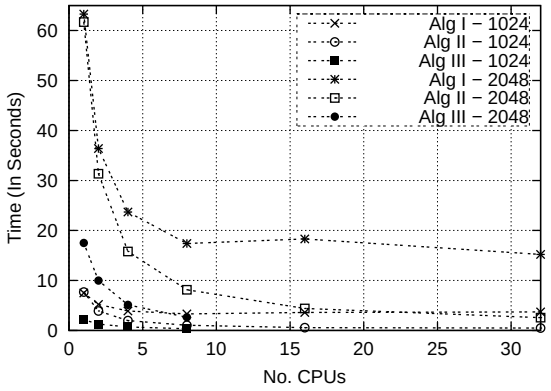


Fig. 4. Curves of Table IV

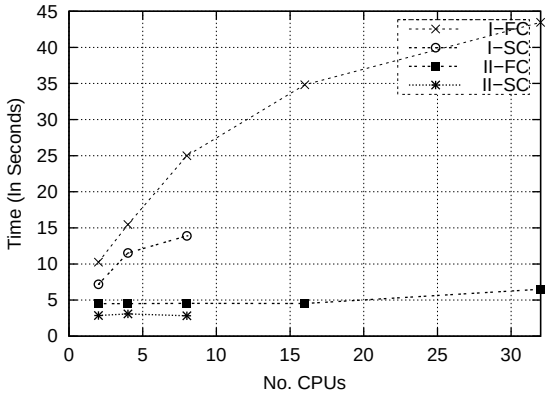


Fig. 5. I-FC: Alves et. al. Algorithm on the First Cluster, I-SC: Alves et. al. Algorithm on the Second Cluster, II-FC: Algorithm 3 on the First Cluster, II-SC: Algorithm 3 on the Second Cluster.

REFERENCES

C. E. R. Alves, E. N. Cáceres, A.A. Castro Jr., S. W. Song, and J.L. Szwarcfiter. Efficient parallel implementation of transitive closure of digraphs. *Lecture Notes in Computer Science*, 2840:126–133, 2003.

S. Baase. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, 2nd edition, 1978.

E. N. Cáceres and C. C. A. Vieira. Revisiting a bsp/cgm transitive closure algorithm. *SBAC*, pages 174–179, 2004.

E. N. Cáceres, S. W. Song, and J.L. Szwarcfiter. A parallel algorithm for transitive closure. *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS 2002, Cambridge, USA, November 4-6*, pages 116–118, 2002.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, 2nd edition, 2001.

F. Dehne. Coarse grained parallel algorithms. *Algorithmica*, 24(3/4):173–426, 1999.

A. Grama, V. Kumar, A. Gupta, and G. Karypis. *An Introduction to Parallel Computing: Design and Analisys of Algorithms*. Addison-Wesley, 2nd edition, 2003.

M. Habib, M.and Morvan and J. Rampom. On the calculation of transitive reduction-closure of orders. *Discrete Mathematics*, 111:289–303, 1993.

J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

R. M. Karp and V. Ramachandran. *Parallel Algorithms for*

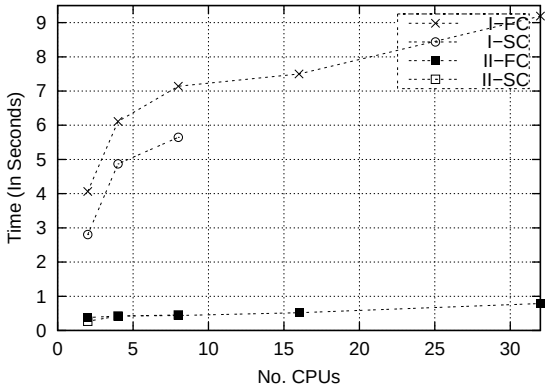


Fig. 6. I-FC: Cáceres and Vieira Algorithm on the First Cluster, I-SC: Cáceres and Vieira Algorithm on the Second Cluster, II-FC: Algorithm4 on the First Cluster, II-SC: Algorithm4 on the Second Cluster....

Shared-Memory Machines - Handbook of Theoretical Computer Science, volume A, chapter 17, pages 869–941. The MIT PRESS/Elsevier, 1990.

A. Pagourtzis, I. Patapov, and W. Rytter. Pvm computation of the transitive closure: The dependency graph approach. *Lectures Notes in Computer Science*, 2131:249–256, 2001.

A. Pagourtzis, I. Patapov, and W. Rytter. Observations on parallel computation of transitive and max-closure problems. *Lectures Notes in Computer Science*, 2474:217–225, 2002.

R. Roy. Transitivité et connexité. *C.R. Acad. Sci. Paris*, 249: 216–218, 1959.

K. Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science*, 58:325–346, 1988.

A. Tiskin. All-pairs shortest paths computation in the bsp model. *Lectures Notes in Computer Science*, 2076:178–189, 2001.

L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103–111, 1990.

S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

AUTHOR BIOGRAPHIES

EDSON N. CÁCERES is a professor of computer science at the Universidade Federal de Mato Grosso do Sul. His research interests are parallel algorithms, cluster and grid computing, and peer-to-peer computing. Cáceres received a DSc from the Universidade Federal do Rio de Janeiro. He is Director of Education of the Brazilian Computer Society and he is a member of the IEEE Computer Society and ACM.

CRISTIANO C.A. VIEIRA is a assistant professor of computer science at the Universidade Federal de Mato Grosso do Sul. His research interests are cluster and grid computing and distance learning. Vieira received a MSc from Universidade Federal de Mato Grosso do Sul.

MODEL OF LOSSY LINKS IN WIRELESS SENSOR NETWORKS

Yelena Chaiko
Department of Industrial Electronics and Electrotechnology
Riga Technical University
Institute of Industrial Electronics and Electrical Drives
1, Kalku Street, LV-1658, Riga, Latvia
E-mail: krivcha@inbox.lv
Viktors Gopejenko
Information System Management Institute (ISMI)
Natural Sciences and Computer Technologies Department
Lomonosova Street, B, Riga, LV-1019, Latvia
E-mail: viktors.gopejenko@isma.lv

KEYWORDS
Model, sensors, wireless sensor networks, communication model.

ABSTRACT

Our first goal is to provide sound foundations for conclusions drawn by extracting relationships between location and communication properties using non-parametric statistical techniques. The objective is to provide a probability density function that completely characterizes the relationship. Furthermore, we study individual link properties and their correlation with respect to common transmitters, receivers and geometrical location.
The second objective is to develop a series of wireless network models that produce networks of arbitrary sizes with realistic properties. We use an iterative improvement-based optimization procedure to generate network instances that are statistically similar to empirically observed networks. We evaluate the accuracy of our conclusions using our models on a set of standard communication tasks, like connectivity maintenance and routing.

INTRODUCTION

The performance of many protocols and localized algorithms for wireless multi-hop sensor networks greatly depend on the underlying communication channel. Hence, to evaluate performance in simulation, we must have an accurate communication model. Until recently, only two approaches have been in widespread use in the sensor network community: unit disk modelling and empirical data traces.

Both approaches present some drawbacks. For example, the unit disk model implies complete correlation between the properties of geometric space and the topology of the network, a property refuted by numerous experiments in actual deployments [1], [2], [3]. When using empirical data traces approach is difficult and expensive to create a large number of large networks that are properly characterized. Therefore, neither probabilistic nor statistical analysis of large networks is feasible. In addition, since a given trace is the result of communication over a specific topology, such a trace does not permit a simulator to reposition nodes. Finally, without validated communication analysis, theoretical analysis is not possible.

In an effort to address this problem, recently there have been a number of efforts to empirically capture communication patterns in wireless sensor networks. In particular, there have been several studies that use different low power, narrow band radio transceivers chipsets to deduce properties of communication links in wireless networks in several environments, such as open space and laboratories. These hybrid models introduce empirically observed factors that modify the communication patterns based on the unit disk communication model.

While these models are a significant step forward with respect to the unit disk model, they are only an initial step in the exploration of the space. These initial models do not capture many important features of communication links in empirically observed networks. For example, they do not address the correlation in communication reception rates between nodes that originated at the same transmitter or differences in the quality of transmitters.

Our goal is to develop accurate simulations of sensor network communication environments that are statistically accurate with respect to several features that impact network protocols and algorithms in real networks. To generate these simulated environments, we construct a set of models that map communication properties such as absolute physical location, relative physical proximity and radio transmission power into probability density functions describing packet reception likelihood. For all of these models, we calculate an interval of confidence. These models not only serve to generate simulated environments, they themselves have lent support to many hypotheses relating to variation in communication link quality [1], [2]. In our study, we do not consider packet losses introduced by multi-user interference (concurrent traffic, contention-based MAC). Nevertheless, our results are useful for three reasons. First, the amount of traffic expected in most application in sensor networks is small, which means either small contention, or in case of highly synchronized events, nodes could be programmed to prevent simultaneous transmissions. Second, they apply directly when using

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

contention free MAC protocols, like pure TDMA or pseudo-TDMA schemes. Finally, they provide a tight upper bound as to what is achievable when using contention-based MAC schemes. The analysis of multi-user interference and temporal properties of the links is part of future work.

```
Conduct exploratory data analysis;
While (interval of confidence>criteria){
Collect new data or define new windows;
Sort all points according to distance;
For (from smallest to largest distance){
Define sliding window for distance;
Apply weight function to distances inside of sliding window;
Sort all points according reception rate;
For(from smallest to largest reception rate){
Define sliding window for reception rate;
Apply weight function to reception rates sliding window;}}
build mapping function;
build normalized mapping function;
establish intervals of confidence;}
```

Figure 1: Pseudo-code of the PDF model generation for two features.

INDIVIDUAL LINK MODELS

In this part we present a statistical approach for building communication link models in wireless multi-hop networks. The goal is to find a statistically sound mapping between two user-specified features that characterize communication links.

Design Guidelines

Our starting task is to analyze the dependency between two properties of wireless networks. We note that exactly the same procedure described below can be used to find the dependency between any two wireless communication features, but for the sake of brevity and clarity, we focus on two specific features: distance and reception rate. The objective is to find the PDF of reception rate for any distance and to calculate intervals of confidence. For example, we could use our model to find that the probability of the reception rate of the link to be 95% at a distance of 25 ft is 0.05 ± 0.000012.

We are guided by three principles, smoothness, compactness, and prediction ability. The validation for adopting these principles is provided by evaluation using resubstitution, which indicates that the derived models have tight interval of confidence and therefore, the statistical model is accurate and the assumptions are justified. The smoothness property states that if two pairs of receivers have very similar distance, their reception rates also often have rather similar probabilistic distribution. In other words, instances of reception rates may be different from one distance to the other, but the underlying reception rate probabilistic distribution is similar. There are two fundamental justifications for this assumption. The first is that at an intuitive level one expects that small changes in one variable (in our case, distance) should have limited impact on the probability distribution of the other parameter (reception rate). In addition, it is important to recognize that both distance and reception rate are subject to errors in measurement that smooth the mapping function.

Quality of the statistical model is ensured through compactness and performance on test cases to measure the prediction ability. There are two sound criteria for any sound statistical model. The first is the Occam principle: the ability to explain a large set of data using a small number of parameters is usually a strong indication that the model will predict well. From a statistical point of view, our goal is to simultaneously have low bias and low variance and therefore low prediction error. Low bias is ensured by preferring models that use fewer parameters. For this task we use Akaike information criteria (AIC). The second criteria is its ability to predict. We scan and alter various parameters in our procedures so long as the adopted parameter values produce a model that withstands standard evaluations of accuracy. Specifically, we use the resubstitution rule, which builds additional models using a variety of randomly selected subsets of the data set. If the resulting models from all data subsets are similar, we conclude that the parameters used were properly selected.

From a technical point of view, when building a model of individual links we have two major difficulties: (i) we do not have enough measurements for each distance of interest, and (ii) for a given distance and given reception rate, we do not have enough collected data samples. We use the kernel smoothing technique to resolve this, and identified that the best performing window had ±10% of the size of the central value and pyramidal shape.

Methodology

Fig. 2 illustrates a scatter plot of distance versus reception rate at radio power for the outdoor case.

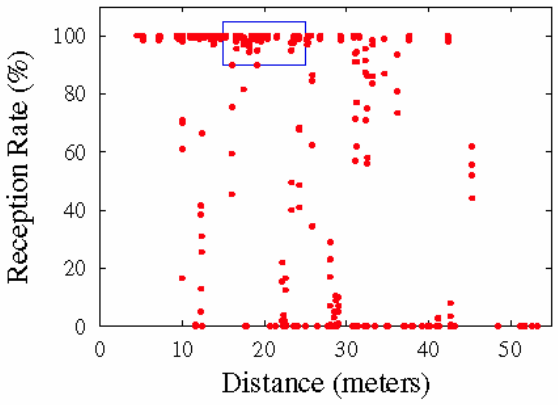


Figure 2: Scatter plot of distance vs.reception rate.

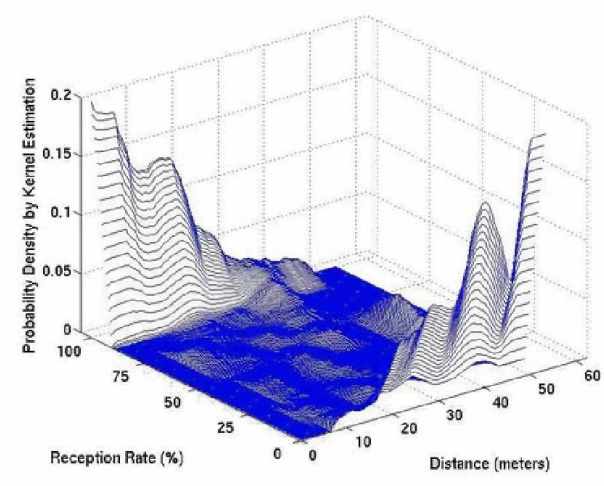


Figure 3: PDF for distance versus reception rate

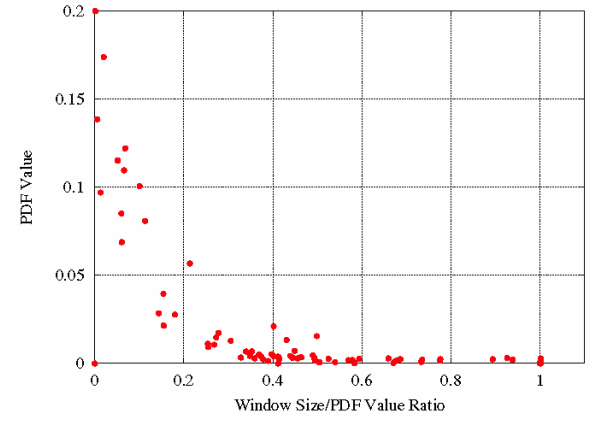


Figure 4: PDF values of the different random points as a function of the confidence interval/PDF value ratio. Outdoor Urban, 90%

We did consider automatic clustering, in particular, self-organizing maps, principal components, independent components and multidimensional scaling, but they did not show intuitive trends.

Phase two consists of three steps shown in Fig. 1 in lines 4-8. In the step shown in line 4, we sort all available data according to distance to identify data points that are similar with respect to this parameter. Next, we use a sliding window for all points which are within a similarity range of a given point (distance). Each point within this range is weighted according to its quantified similarity to a given point. For each distance of interest we also build another system of sliding windows this time along the y-axis corresponding to the reception rate. The points within the window are weighted as the product of the weight factor of both the distance window and the reception rate window.

Once the first eleven lines of the pseudo-code are executed we have enough information to build a PDF that indicates how likely a particular reception rate is for a given distance. For this task, following compactness principle, we used quadratic least linear squares fitting for a particular pair of distance and reception rate.

Once the model is built, the next step is PDF normalization that ensures that for a given distance the integral of the function below the PDF mapping function is equal to one. Fig. 3 illustrates how the normalized reception rate PDF changes with respect to distance.

Table 1: Global evaluation results

Environment	Conf. Level	H.L.PDF Value	Conf. Intervals
Indoor	90	0.997627	±0.325969
Outdoor	90	1.064365	±0.381719
Indoor	95	1.023886	±0.723887
Outdoor	95	1.022372	±0.691752

Evaluation.

The final step of our procedure is the evaluation of the quality of the developed statistical model for the PDF. The evaluation procedure itself has three components: Monte Carlo sampling, resubstitution, and establishment of interval of confidence. Monte Carlo sampling selects k (in our experimentation we use 200) randomly selected pairs of distance and reception rate points.

Resubstitution is the process where a statistical model is built using the exact same procedure (same kernel window scope and weight function) on randomly selected subsets of data. Specifically, in our simulations, we select 70% of the available data to build a model on each resubstitution run. For each resubstitution run we record the value of the PDF function at each of the k selected points. After conducting m resubstitution runs (in our experimentation m was 100), we are ready to establish an interval of confidence for our statistical PDF model. This is performed in two stages. We first establish an interval of confidence for each point individually, and then by combining information from all local interval of confidence we establish a global interval of confidence. Fig. 4 shows the relationship between the different confidence intervals for each random sample tuple (reception rate and distance) and the highest likelihood PDF value for different confidence levels. Each point in the graphs show the highest likelihood PDF value with its confidence interval. For example, the top left point in the graph of Fig. 4 corresponds to sample point of distance 52 meters, reception rate 0% with highest likelihood PDF value of 0.2 ± 0.0001953 with confidence level of 90%. The final step of resubstitution is to build a global measure of the model's accuracy. To build a global interval of confidence we use the following procedure. First, for each separate point in k, we use the highest likelihood PDF value and normalize all other values against this value. After that, we combine all data from all sampling points into one set of the size k x m. Finally, we calculate the confidence intervals of the normalized global array. Table 1 shows the overall interval of confidence for indoors and outdoors with different confidence levels. In general, the global highest likelihood PDF values are centered around one,

which is a good sign of the statistical soundness of the model.

PROPERTIES OF INDIVIDUAL LINKS

At the highest level of consideration the features can be classified into two groups: physical properties of the network, and communication features of the network. Physical properties include distance, direction as a function of angle with respect to reference direction, and areas. Communication properties include reception rate between receiver A and transmitter B, asymmetry in communication which refers to the absolute difference in reception rates between a pair of nodes (transmitter A → receiver B and transmitter B → receiver A), and temporal variation of reception rate between receiver A and transmitter B.

We have analyzed two types of mapping functions between properties. The first is the established one-way mapping relationship between a given structural property and a targeted communication feature. The second analyzes the one-way dependency between two communication features. We once again emphasize that our goal is to not only establish the most likely value of one property for a given value of another property, but also to obtain probability distribution functions for all expected values of the second property for a given value of the first feature. We have studied the following pairs of properties.

Dependency of reception rate as a function of distance. This property is selected mainly because there is a wide consensus that distance significantly impacts reception rate.

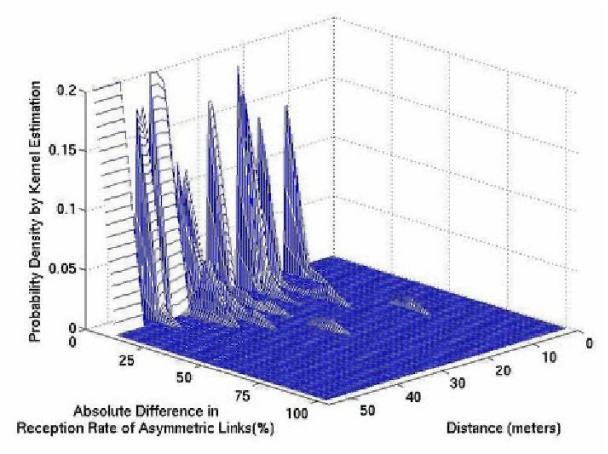
Dependency of asymmetric reception rate as a function of distance. Note that in the previous case we assumed that reception rate between transmitter A to receiver B is the same as from transmitter B to receiver A. Our goal is to quantitatively capture how frequently there is asymmetry in reception rates as a function of distance.

Dependency of asymmetric reception rate as a function of reception rate. This property studies functional dependencies between two communication properties. Our goal is to identify if it is more likely that high asymmetry happens when links have high, low, or medium reception rates. For example, we are interested if it is more likely to have a pair of nodes with reception rates of 95% and 75%, or with 30% and 10% reception rates.

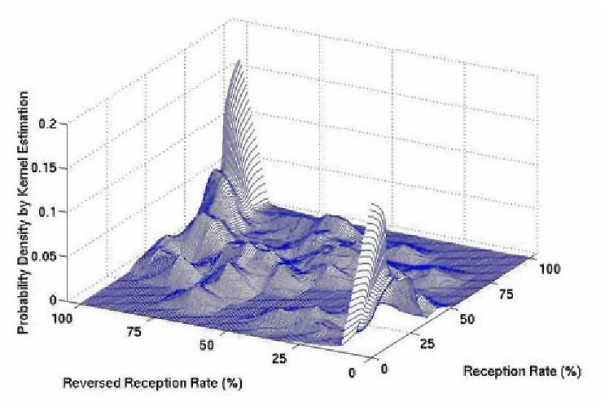
Dependency of reception rate standard deviation as a function of the average reception rate. The final property studies temporal dependencies between two communication properties. Our goal is to quantitatively capture this relationship and provide some initial results on how this property affects the link estimation algorithms used for online quality estimation.

In addition to the listed properties, we also studied link quality dependency on angle, but were not able to identify any interesting patterns with significantly strong intervals of confidence.

In paragraph Individual link models, Fig. 3 we have illustrated how the reception rate changes as a function of distance. Our results confirm the findings of several studies in the literature that show that there is a significant percentage of the radio range where links are highly variable, with similar probabilities of having very high or low reception rates. Even for very short distances, the probability of having very low reception rate links is not zero, and it starts growing fast as distance increases. More importantly, the average and standard deviation values of reception rate are insufficient parameters to model reception rate as a function of distance. While the average reception rate is around 50% in this case, most of the links have either very high or low reception rates.



5(a) Asymmetric links vs distance



5(b) Asymmetric links vs reception rate

Figures 5: PDFs for asymmetric links features

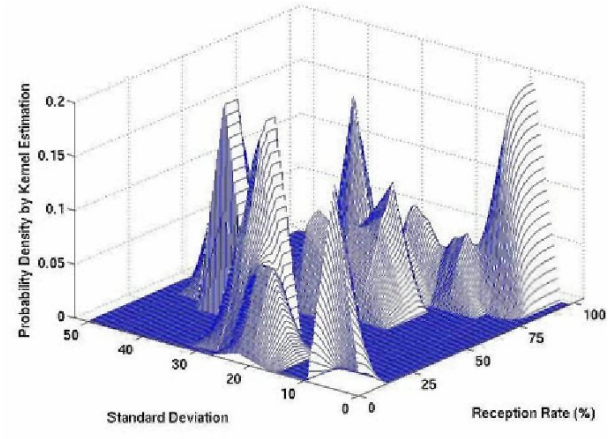
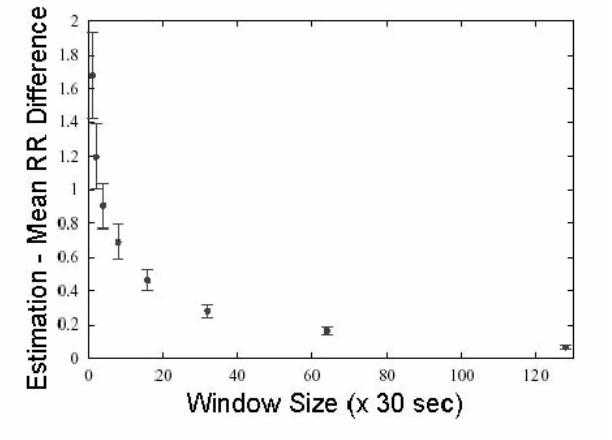
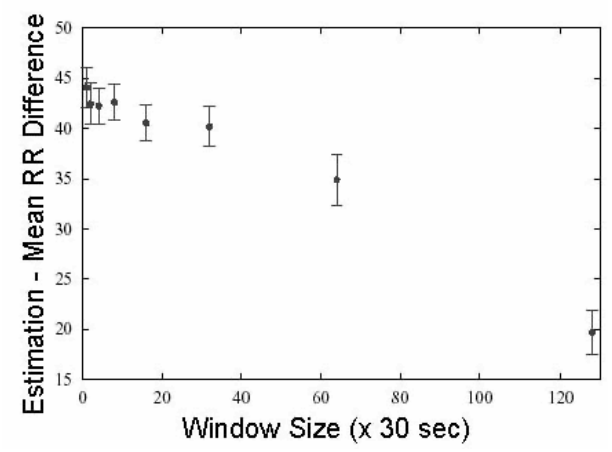


Figure 6: PDF for temporal variation as a function of the reception rate.



7 (a) High Reception Rate



7(b) Med. Reception Rate

Figures 7:Time series for on-line link quality estimation.

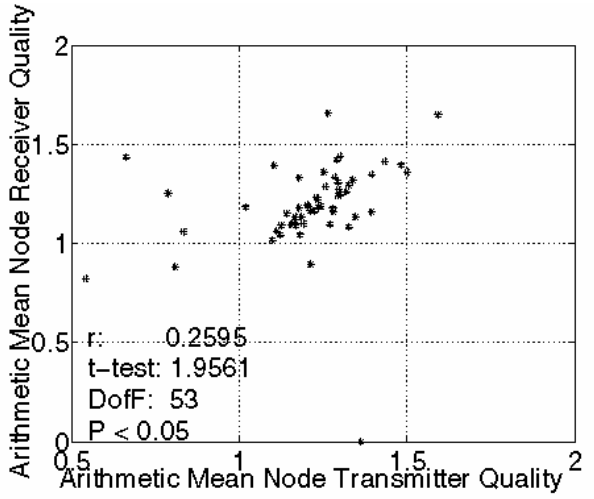
Figs. 5(a) and 5(b) show the PDF of how asymmetric reception rate depends on distance and average reception rate. Fig. 5(a) shows that there is no clear correlation between link asymmetries and distance. Fig. 5(b) shows an interesting pattern; links with very high or very low reception rates tend to be highly symmetrical, as it can be observed by the two peaks in the PDFs. Links with medium reception rate tend to be much less symmetrical.

Fig. 6 shows the temporal variability of the links as a function of the reception rate. We clearly see that links with very low or very high reception rates tend to be more stable over time (smaller standard deviation), while the links with intermediate values of reception rate tend to be more unstable (higher standard deviation).

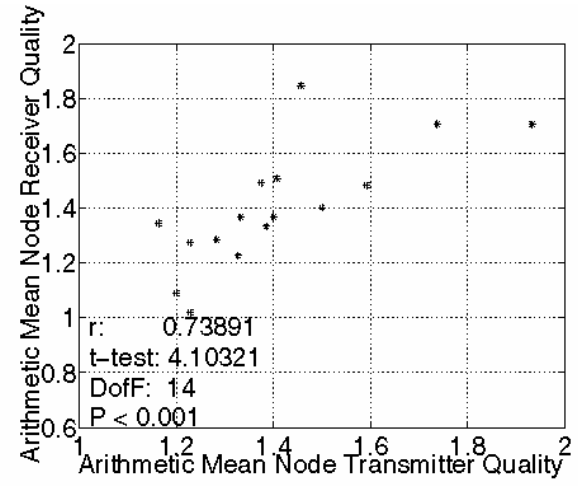
From our data, we observe that while the quantitative values of the PDFs for different conditions were not the same, the PDFs generated were qualitatively similar in most cases. For example, the PDF shown in Fig. 3 was qualitatively similar across all three environments tested. We have left the statistical analysis of the differences between the different conditions to get statistical sound conclusions for future work because it requires additional experiments.

One interesting question we wanted to answer is how long a node needs to measure the communication channel in order to get an accurate estimate of reception rate with a certain confidence interval. This has a profound impact in the design of algorithms for topology control that need to measure the channel as little as possible in order to save energy by periodically turning the radio off. To evaluate this, we took long time series of reception rate data, and picked k window sizes. For each window size, we took p (set to 100) initial random points of measurements from the time series, generating a reception rate estimate for each p using only a window of size k (ranging from 30 seconds to 64 minutes) of data from the starting point. Then we compare the absolute difference between each of the p × k estimates with the absolute reception rate calculated using the entire time series of data. Fig. 8 shows the results of the previous analysis on two qualitatively different type of links. Fig. 7(a) shows that links with very high reception rate need very short window sizes to get an accurate estimate of the reception rate, and they converge quite fast to an accurate estimate (low reception rate links show similar behavior). Fig. 7(b) shows that links with intermediate reception rates take much larger window sizes to converge to accurate estimate values. We have left for future work the issue of optimal on-line link characterization using statistical methods.

From the spatial, asymmetrical, and temporal properties presented in Figs. 3, 5(b) and 6 we can see an interesting pattern that has emerged. For a large range of distances there is a low but non-zero probability of links with medium reception rates. These links are also the ones that present the most highly asymmetrical and temporal variability properties. We believe these links may introduce serious stability and convergence problems for several routing algorithms, and it might be useful to design mechanisms to detect these types of links and filter them out.



9(c) Mean of Quality Correlation (per node): Outdoor



9(d) Mean of Quality Correlation (per node): Indoor

Figures 9: PDF for Normalized Transmitter and Receiver Quality and Correlation

Another interesting point is that reliable, highly symmetrical and stable links exist even at long distances (although with low probability). It is important to detect and take advantage of those long distance/high quality links in order to minimize packet transmission in a multihop setting. Online detection and use of these type of links could affect algorithm design. (e.g. by minimizing energy consumption or end-to-end hop count).

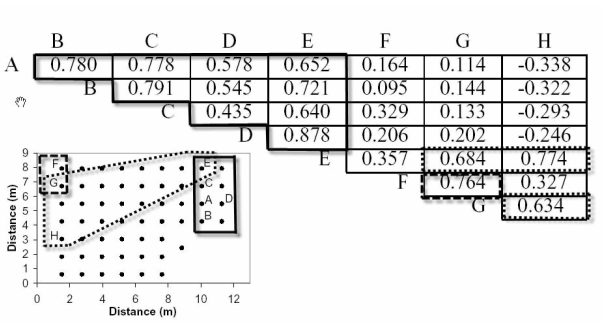
GROUP LINK PROPERTIES

Group link properties are joint properties of related subsets of links. These links may be links that originate from the same transmitter or received by the same receiver, processed by the same radio, or communicated by nodes that are geometrically close. These properties are of crucial importance for any analysis and answer the frequently asked fundamental questions about reasons for particular behavior of communication patterns. These questions include whether the performance of a particular node as a transmitter mainly depends on the quality of its radio or its geometric position. Another frequently asked question is whether asymmetry is a consequence of different radio properties between two nodes or their location. However, with the exception of the property which examines pairs of links between two nodes, group link properties have been rarely studied due to their perceived complexity.

The first question we answer is to what extent the quality of transmitters and receivers on different nodes is uniform. We normalized the quality of each link at each node versus the average link quality at the corresponding distance in terms of reception rate. After that, we calculated the geometric mean of all links that originate or end at a particular node. For the reception quality, we decided to use the geometric mean instead of arithmetic in order to avoid too high an impact from a few exceptionally strong links. For example, if a node has one link that is 5 time better than average and 5 links that are 5 times worse than average, the arithmetic mean would still indicate that the nodes have links of superior quality, which is obviously not the case. When there are not outliers, the arithmetic mean is preferred.

Table 3: Correlation of all pairs for indoor and uotdoor

	Outdoor			Indoor		
	r	t-test	DofF	r	t-test	DofF
TX	-0.004	0.121	887	0.0592	11.824	39770
RX	0.012	0.370	885	0.0590	11.859	40183



Figures 10: Covariance Matrix and Layout for Indoor Experiments.

Figs.9(c) and 9(d) show a summary of our results. We analyzed both indoor and outdoor data using arithmetic mean. We calculated both arithmetic and geometric mean correlations, but due to the lack of outliers in the data, we preferred to use the arithmetic mean. All studies indicate that there is a positive correlation of transmitting and receiving capability of the nodes, and the probability of this result being accidental is low (lower than 0.1% in the indoor case). The linear correlation factor values are different depending on the environment, being much higher for the indoor case.

Once we conclude that some nodes are much better transmitters or receivers than other nodes, the natural question is to what extent they are uniformly better transmitters or receivers with respect to all their links. In order to answer this question we calculated the correlation between all transmitting (receiving) links related to the same node. Table III shows the correlation value r, the t-test value and the degree of freedom (DofF). For both indoor and outdoor environments we see essentially very small or no correlation with very high probability (the probability of this result being accidental is lower than 0.1% for the indoor case). This essentially means that no node has perfectly good links to all other nodes in some distance range, and even the best nodes have average or very poor links. In addition, almost all nodes have good links to some neighbor in the same distance range.

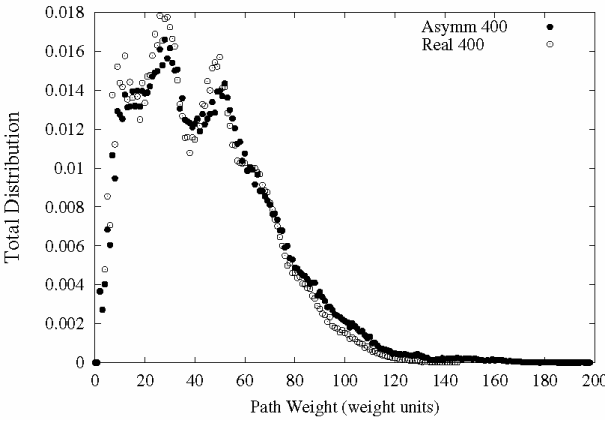
The last question we would like to answer is whether there
are subsets of nodes that communicate well with each other while communicating at significantly lower levels with other nodes in the network. Fig. 10 shows the covariance matrix for 9 nodes in the indoor environment. We clearly see that nodes A, B, C, D and E form one group, nodes E, G and H another, and F and G, the third group. All nodes in these groups are highly correlated in terms of normalized communication with respect to other nodes. The data was obtained in the following way. For each node we sorted in decreasing order the quality of its links to other nodes. After that, for each pair of nodes, we found a subset of corresponding receivers that hear both nodes, and eventually found rank correlation for these two lists. As part of the table indicates, very often the correlation between two nodes is rather high, close to positive 1 or very low close to -1. The Spearman t-test indicates that all covariance values have probability of accidentally happen well below 0.1%. In other words, group of nodes in a particular distance range can communicate to each other significantly better than other group of nodes in the same distance range. Identification of these groups of nodes could be important for tree-based routing algorithms; it would be convenient that at least one node in each of these groups join the tree since it could communicate better to the other nodes in the group than any other node.

WIRELESS NETWORK GENERATORS

Using the knowledge gained from analysis of single and multiple link properties, we have built a series of wireless multi-hop network instance generators to be used in simulation environments. We analized three models, increasing in complexity, which create communication links for an arbitrary network that are statistically similar to observed networks. The basic model assigns communication links based solely on the relationship between reception rate and distance. To build the more complex models, we introduce an iterative improvement-based procedure for creating communication links which abide by multiple link properties. The starting point for all models is the generation of a user specified number of nodes in the given area, with specific locations or a particular distribution.

Table 4: Comparison of four statistical models using Floyd-Warshall all pair shortest path algorithm

	Unit	Unit real	Prob.	Prob-real
min	2	2	2.0079	2.0079
max	26	20.0569	41.881	41.881
ave	6.87574	5.78918	14.687	14.687
	Asymm	Asymm real	Statistical	Statistical real
min	2.00188	2.00188	2.00002	2.00002
max	45.9964	44.1535	42.99	42.9285
ave	14.8176	14.6217	14.6991	14.0028



Figures 10: Similarity between path weights in large networks.

We compared using the perturbation-based method four models: unit disk model, probabilistic disk model, asymmetric probabilistic disk model, and a non-parametric statistical model. For this purpose we compare the length of all-pairs shortest paths for an instance with 400 nodes. Table 4 provides a summary for the length of the minimal, maximal, and average path. Note, that all three newly developed models, and in particular the non-parametric statistical one, are much more statistically sound.

DESIGN CONSIDERATION AND CONCLUSION

From the conceptual point of view, the first important observation is that the distribution of lossy links can

greatly affect routing algorithms based on geometric concepts. For example, all local avoidance approaches that reduce the routing problem to traversal on Gabriel or local neighborhood graphs may no longer be applicable in practice. Another, possibly more impacting ramification is that no deterministic method can be used to guarantee packet delivery in stateless routing protocols. This is justified by the small but non-zero probability of having links with very small or close to zero reception rate even at very small distances (Fig. 2). The third major conceptual change is that there is a strong benefit of observing at least some percentage of links on-line. This is because some of the most effective links in terms of metrics of travel distance versus required number of messages are links that have a reception rate between 40-60%. In addition, we can observe from Figs. 2, 5(b) and 6 that it is perfectly possible to find high reception rate links that are stable and highly symmetrical that cover medium to long distances.

The complex and correlated nature of links implies that newly developed routing protocols should be simulated for much longer periods of time in order to ensure that overall they perform well. The existence of superior nodes in terms of both transmitters and receivers capabilities implies that fairness will become one of the major issues for any routing, multicast, and broadcasting approach, because all of these protocols have a tendency to disproportionately use a subset of nodes. The statistically demonstrated space correlation will also greatly impact the development of routing protocols, as well as power management techniques. For example, since nodes are naturally clustered in subsets that efficiently communicate with each other and poorly with the rest of the network, it will be important that power management strategies, simultaneously turn down or up the majority of the nodes in one of such subsets. Furthermore, clustering techniques might be even more efficient than in networks modeled with the unit disk communication model.

In summary, we have developed a set of non-parametric statistical models for characterizing links in wireless sensor networks. The models are the basis for new generators of wireless networks to be used in simulations that are statistically similar to deployed networks. The insight gained while building these models has helped identifying future directions for developers of protocols and localized algorithms for wireless sensor networks.

REFERENCES

A. Cerpa, N. Busek, and D. Estrin, "SCALE: A tool for simple connectivity assessment in lossy environments," CENS, UCLA, Tech. Rep. 0021, Sep 5 2003.

D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: An experimental study of low-power wireless sensor networks," CENS, UCLA and IRL, UCB, Tech. Rep. 02-0013, February 2002.

Y. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in Proceedings of ACM Sensys 2003. Los Angeles, CA, USA: ACM, Nov 5–7 2003, pp. 1–13.

T. S. Rappaport, Wireless Communication: Principles and Practice. Prentice Hall, 2000.

A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in Proceedings of ACM Sensys 2003. Los Angeles, CA, USA: ACM, Nov 5–7 2003, pp. 14–27.

G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Impact of radio irregularity on wireless sensor networks," in International Conference on Mobile Systems, Applications and Services, 2004, pp. 125–138.

D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in Proceedings of ACM SIGCOMM 2004. Portland, OR, USA: ACM, Aug 30–Sep 3 2004.

J. E. Gentle, W. Hardle, and Y. Mori, Handbook of Computational Statistics, Concept and Methods. Springer-Verlag, 2004.

AUTHOR BIOGRAPHIES

YELENA CHAIKO In 2004 – Doctor of Science in Telecommunication " Mathematical models of radio wave propagation in woodland for mobile communication systems". From 2006.01.10. Riga Technical University as head researcher, Institute of Industrial Electronics and Electrical Drives. Research interests: Telecommunication systems, Electronics. Riga Technical University, 1, Kalku Street, LV-1658, Riga, Latvia, e-mail: krivcha@inbox.lv .

VIKTORS GOPEJENKO is an associated professor of the Natural Sciences and Computer Technologies Department of Information System Management Institute (ISMI) Riga, Latvia. He holds a Dr. Sc. (engineer) from the Moscow Civil Aviation Engineer Institute in 1987. His research fields include mathematical and computer modeling, digital signal processing. His e-mail address is: viktors.gopejenko@isma.lv, Web-page – www.isma.lv

.

Workshop onSecurity and High Performance Computing Systems

An Architecture for Distributed Dictionary Attacks to OpenPGP Secret Keyrings

Massimo Bernaschi
IAC-CNR
Viale del Policlinico, 137, Rome, Italy
massimo@iac.rm.cnr.it

Mauro Bisson
CASPUR
Via dei Tizii, 6, Rome, Italy
m.bisson@caspur.it

Emanuele Gabrielli
Dip. di Informatica e Scienze dell'Informazione
Università di Genova, Italy
gabrielli@disi.unige.it

Simone Tacconi
Ministero dell'Interno
Servizio Polizia Postale e delle Comunicazioni
simone.tacconi@interno.it

Abstract

We describe a distributed computing platform to lead large scale dictionary attacks against cryptosystems compliant to OpenPGP standard. Moreover, we describe a simplified mechanism to quickly test passphrases that might protect a specified private key. *Only passphrases that pass this test complete the full (much more time consuming) validation procedure. This approach greatly reduces the time required to test a set of possible passphrases.*

1 Introduction

A dictionary attack is a technique for defeating a cryptographic system by searching its decryption key or password/passphrase in a list of words or combinations of these words. Although it is widely accepted that the main factor for the success of a dictionary attack is the choice of a suitable list of possible words, the efficiency and reliability of the platform used for the attack may become critical factors as well. Hereafter, we present a distributed architecture for performing dictionary attacks that can exploit resources available in local/wide area networks (in P2P style) by hiding all details of communication among participating nodes. As an example of possible cryptographic challenge for which the platform can be used, we selected the decryption of the private *keyring* of the GnuPG software package. From this viewpoint, the present work can be considered a replacement and an extension of *pgpcrack* (that is no longer available), an utility used for cracking PGP encrypted files. Note that the structure of the OpenPGP *secring* is much more complex with respect to the original PGP. To the best of our knowledge, no equivalent *fast* cracking system exists for OpenPGP. Other scalable distributed cracking systems were proposed in [3] and [9]. Due to the lack of space we can not present a detailed comparison, but we just mention that our system pays much more attention to reliability and portability issues than the cited systems. The paper is organized as follows: Section 2 describes the features of OpenPGP, the standard to which GnuPG makes reference; Section 3 describes our approach to the attack of the GnuPG *keyring*; Section 4 introduces the architecture we propose for the distributed attack; Section 5 gives some information about the current implementation; in Section 6 we present some preliminary results and, finally, Section 7 concludes with future perspectives of this activity.

2 OpenPGP Standard

OpenPGP is a widely used standard for encryption and authentication of email messages. It is defined by the OpenPGP Working Group in the Internet Engineering Task Force (IETF) Proposed Standard RFC 2440 [5]. OpenPGP derives from PGP (Pretty Good Privacy), a software package created by Phil Zimmermann in the beginning of nineties. GnuPG [2] is a well-known example of software package compliant to OpenPGP standard available in the public domain. New commercial versions of PGP are also compliant to OpenPGP standard.

The OpenPGP standard adopts a hybrid cryptographic scheme. For instance, message encryption uses both symmetric and asymmetric key encryption algorithms. The sender uses the recipient's public key to encrypt a shared key (i.e. a secret key) for a symmetric algorithm. That key is used to encrypt the plaintext of the message. The recipient of a PGP encrypted message decrypts it using the session key for a symmetric algorithm. The session key is included in the message in encrypted form and it is decrypted in turn by using the recipient's private key. These keys are stored in

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

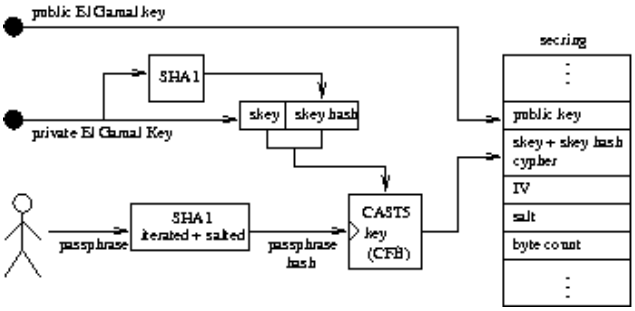


Figure 1. OpenPGP *keyring*

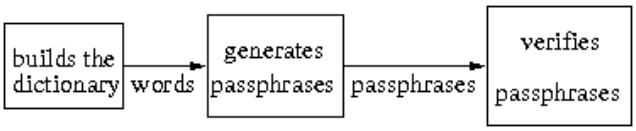


Figure 2. The three phases of the attack

two separate data structures, called "keyrings": private keys in the private keyring, public keys in the public keyring. Every keyring is a list of records, each of which associated to a different key. In order to prevent disclosures, private keys are encrypted with a symmetric algorithm, by using a hash of a user-specified passphrase as secret key. For what concerns GnuPG, as shown in Figure 1, the asymmetric encryption algorithm is El Gamal [7], the hash algorithm is SHA1 [6] and the symmetric encryption is CAST5 [4], used in CFB mode [5].

3 Attack Strategy

One of the most critical issues regarding OpenPGP security is the secrecy of passphrases protecting private keys. The knowledge (by any means achieved) of the passphrase allows a malicious user to perform critical operations as signature and decryption of messages belonging to the legitimate owner of the passphrase. For this reason, the goal of the attack is to find the passphrase, starting from a private keyring in OpenPGP format. The attack is divided in three phases, each of which receives as input the output of the preceding step, as shown in Figure 2.

The first phase is devoted to build the dictionary used for passphrases generation that represents the second phase. The third phase consists of the test of every generated passphrase.

3.1 Dictionary Compilation Phase

In this phase, the basic dictionary is created starting from a set of text files. This a pretty simple procedure: each different word is placed in the list that constitutes the dictionary. In order to increase chances of success, the content of these text files should contain information somehow related to the legitimate owner of the passphrase under attack. This process is depicted in Figure 3.

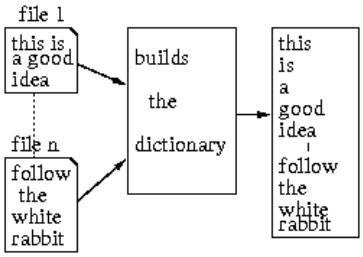


Figure 3. The first phase: the build of the dictionary

3.2 Passphrase Generation Phase

This second phase produces a list of passphrases by applying a set of generation rules to all words found in the dictionary. Every rule involves the current word and a chosen number of subsequent words and allows for the generation of passphrases, by performing permutations of the order of words and/or substitutions of single characters. In this way, the obtained passphrases are reasonably compliant with rules of natural language.

For instance, if we apply rules that involve a word and four subsequent words to generate passphrases with a length ranging from one to five words, for each word in the dictionary we obtain 39 possible passphrases:

- the current word as in the dictionary, then the same word with all lower case letters and all upper case letters (3 passphrases).
- the current word and the following one, taken in the original order and in the reverse order, with all lower case letters, all upper case letters and the unmodified case (6 passphrases).
- all possible permutations of the current word and the two subsequent words, with all lower case letters and all upper case letters (18 passphrases).
- the current word and the three subsequent words, taken with the order and the case in the dictionary and in reverse order, with all lower case letters and all upper case letters (6 passphrases).
- the current word and the four subsequent, taken with the order and the case in the dictionary and in reverse order, with all lower case letters and all upper case letters (6 passphrases).

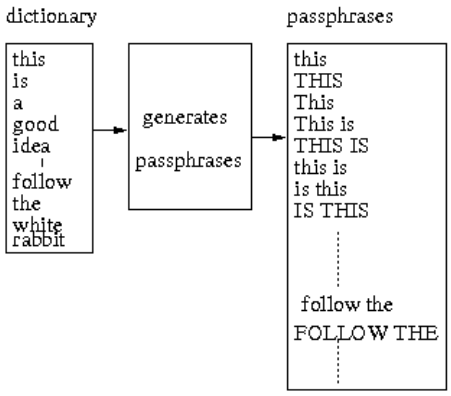


Figure 4. The second phase: generation of passphrases.

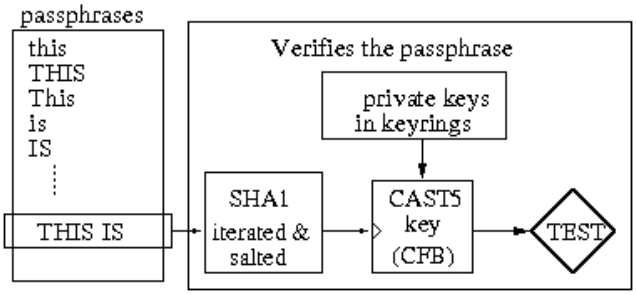


Figure 5. The third phase: verification of the passphrase.

Note that in the generation of passphrases with four and five words, some permutations are not considered, since they yield sequences unlikely for human memorization. The generation phase is depicted in Figure 4.

3.3 Passphrase Verification Phase

This phase is the *core* of the attack and the most expensive from the computational point of view. Each passphrase generated in the previous phase is checked by following an *incremental* approach aimed at minimizing the cost of the controls required by the OpenPGP standard. For this reason, a symmetric key for CAST5 algorithm is derived from every passphrase, by applying the SHA1 algorithm in iterated and salted mode. Such a key is used to try a decryption of encrypted components relating to private key. This process is represented in Figure 5.

In order to check whether the passphrase under test is the right one, it is necessary to verify the plaintext obtained from the decryption procedure. This operation is performed taking into account how the OpenPGP standard represents components relating to private keys in keyrings.

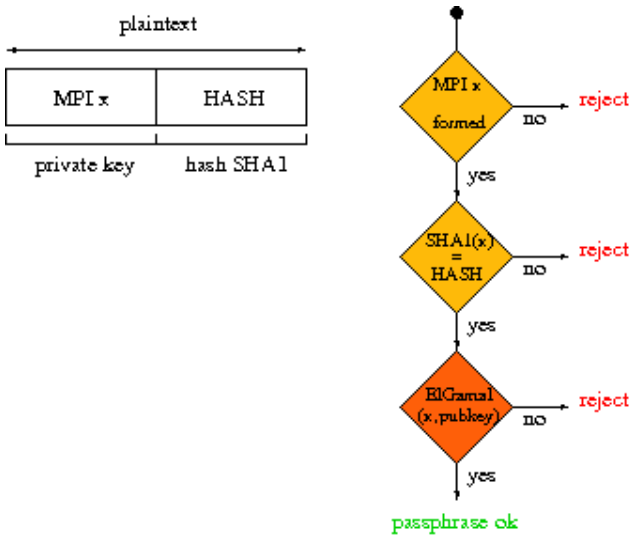


Figure 6. Validation test

As shown in Figure 6, a private key is represented as a Multi Precision Integer (MPI), followed by its hash, computed with SHA1. For this reason, in the first step we verify if the left part of plaintext is a well-formed MPI. In case of success, we double check whether the result of SHA1 applied to the MPI matches with the hash found in the plaintext. Only for those passphrases that pass this second test, we control the fulfillment of the algebraic relationship that should be between the MPI and the corresponding public key. If this final check is successful, then the passphrase under test is the correct one. Note that the first two controls have a low computational cost but they may produce *false positives*. The last control is *exact* but it is very expensive from the computational viewpoint. GnuPG does not carry out the first control that is already very selective. By following this multi-step procedure our validation test is much more effective.

4 Distributed Architecture

The attack described in the previous section has been deployed over a loosely coupled distributed architecture. The three phases of the attack are scattered over the nodes of this network. There is a main node and two different groups of peers that share their computational resources.

4.1 General Requirements

Since this network has been conceived to work with heterogeneous systems in a geographic context, the proposed architecture guarantees the following requirements:

scalability: the number of network nodes can be increased,

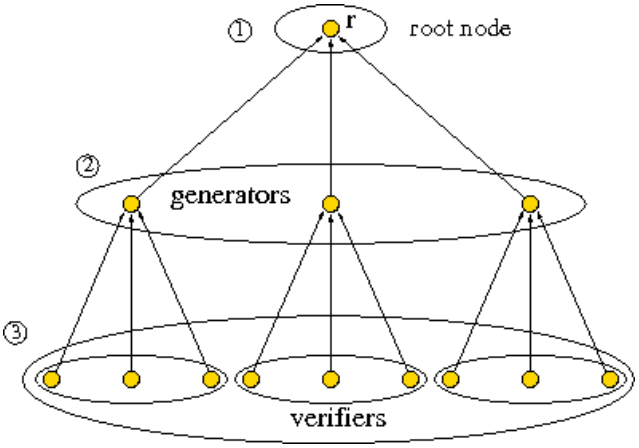


Figure 7. Nodes organization.

augmenting the overall computational power.

load balancing: the computational load must be distributed among nodes in proportion to their capabilities, so that to avoid local starvation.

flexibility: since the availability of each node in the network is aleatory, the architecture must be able to adapt itself to variations of available resources by changing the distribution of charge.

fault tolerance: possible failures of a node must not subvert the overall computation, thus the system must able to re-assign the workload and to resume local computation.

4.2 Overall Organization

The proposed architecture consists of three levels, each of which implements a specific phase of attack, as represented in Figure 7. Each level receives information from the upper level, elaborates them and then supplies the lower level.

The first level is constituted by a single "root" node, denoted as r, that is responsible for the compilation of the dictionary. The second level consists of a variable number of nodes, named "generators" and denoted as g, that form the "generation network", indicated as G. Such a network is devoted to generation of passphrases starting from the dictionary compiled in the first phase by the "root" node. The third level consists of a variable number of nodes, named "verifiers" and denoted as v, that form the "verification network". Such a network is in charge of verifying whether any of the generated passphrases decrypt the private key given in input. Node r and the sets of nodes G and V form the network system $\sum =< r, G, V >$.

System $\sum$ has a tree-like topology where generator nodes play the role of children of root node r. Verifier nodes v are divided in groups, each of which is assigned to a generator node g, as depicted in Figure 7. Each node acts as client with respect to the father node and as server with respect to the child node.

Every node performs a specific task:

- root node r compiles the dictionary D, divides it in partitions $P_i(D)$ and assigns the i^{th} partition to the generator node g_i;
- each generator node g_i extracts from $P_i(D)$ a list of passphrases L and divides it in partitions $P_j(L)$. Every partition $P_j(L)$ is assigned to a verifier node v_j^i (where the superscript i indicates that v_j is a child of g_i);
- each verifier node v_j^i checks all passphrases in the assigned partition $P_j(L)$ with respect to the private key provided in input.

This model of interaction, represented in Figure 8, makes easier to achieve a reasonable load-balancing by assigning more work to groups with more verifier nodes. Every node of the network needs to know only the identifier of its father node, of the "root" node and of all its child nodes (if any), in order to communicate with them. Moreover, for each of its child nodes, a father node checks the status of available resources and stores the last messages sent to it. Information stored in a node are maintained until child nodes do not confirm the completion of operations assigned to them. Note that child nodes never communicate each other.

Communication occurs by means of message exchanges that require receiver's confirmation. A node only accepts messages coming from the father, its children and, possibly, the root. Messages can be grouped as follows:

task messages: used to exchange information about the attack.

maintenance messages: used for handling asynchronous events of the network.

heart-beating messages: aimed at detecting failures and sending information about available resources.

4.3 System Life-cycle

An instance of the system begins with just the root node. As new nodes join the network to participate in the attack, (this is done by sending a message to the root node), generation and verification networks are populated. A new node is assigned to the generation network if there are no generator nodes (this is the typical situation in the beginning), or if all existing generator nodes serve the maximum number

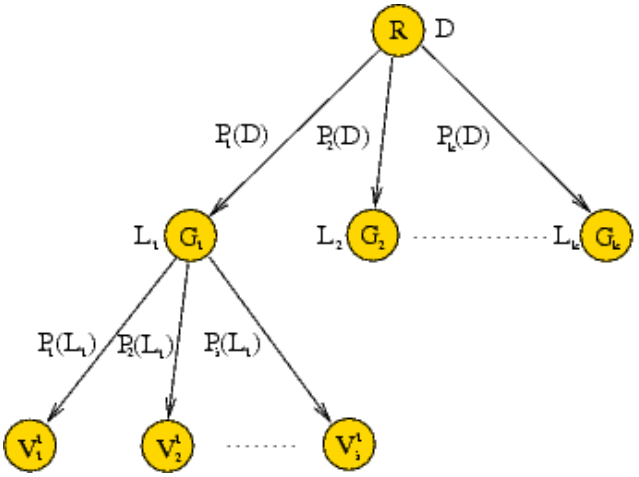


Figure 8. Interaction among nodes

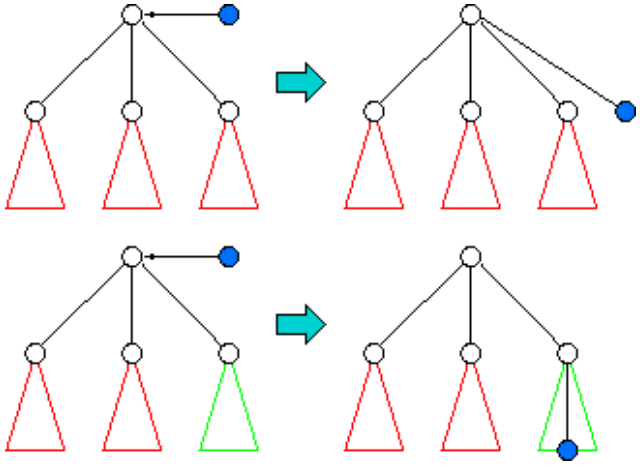


Figure 9. Propagation scheme

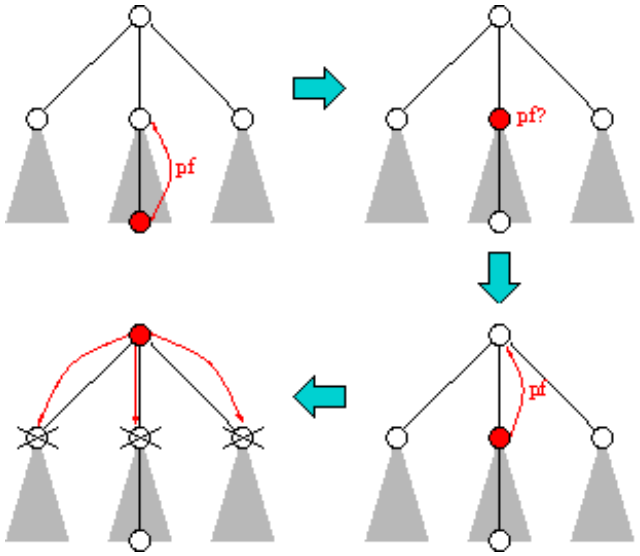


Figure 10. Shutdown scheme

of verifier nodes (this maximum number can be tuned at run time). Otherwise, the new peer becomes a verifier node and it is assigned as child to the generator node having the lowest number of children. The expansion model of the system is shown in Figure 9.

An instance of the system ends when the correct passphrase for the given private key is found. The verifier node on which a candidate passphrase passes successfully the first two controls described in section 3.3 sends it to its father generator node. This node performs further controls (the final test described in section 3.3) and, on success, then forwards the passphrase to the root node that, as a consequence, stops the system. This process is depicted in Figure 10. For what concerns the single nodes, every peer can be in one of the following states:

running: the node is performing its own task;

serving: the node is executing the assigned task and performing a maintenance operation that involves one or more child nodes;

stopped: the node is not executing a task because it is involved in a maintenance operation launched by its father node or by itself;

The root node can be in either running or serving state, a generator node can be in running, serving or stopped state, a verifier node can be in either running or stopping state. State transitions occur when a message is received, or as a consequence of a local event.

Each node is able to produce local events that are handled by executing maintenance operations. Nodes generate events that are compatible with their current state. An event triggers a transition in a state where the corresponding maintenance operation must be executed. Three kinds of events are possible:

soft-quitting (SQ): produced when a node explicitly leaves out the system;

hard-quitting (HQ): generated when a node detects an unexpected quitting of a child, for example due to a child failure.

swapping (SW): event that occurs when a node exchanges its role with a child.

Each node manages its soft-quitting related operations and hard-quitting related operations of its children. Verifier nodes, since do not have children, do not need to manage hard-quitting and swapping events. Finally, the root node can not swap its role with a child.

If the root node quits the network, the entire instance of the system halts. When a failure (HQ operation) occurs

in a generator node, the root appoints one of the orphan verifier nodes as new generator node for the remaining orphans nodes and assigns to it left pending partitions by the failed generator node. When a generator node wants to quit the system (SQ operation), it elects a substitute, choosing it among its child (verifier) nodes, and supplies to it all the information required to complete the task. Finally, the outgoing node informs the root node and quits.

When a failure occurs in a verifier node (HQ operation), the father generator node forwards to other child nodes the pending list of passphrases previously assigned to the broken node and informs the root. When a verifier node wants to quit the system (SQ operation), it informs its father generator node about the number of checked passphrases in the pending list. The generator node then supplies residual passphrases to its other child verifier nodes and informs the root note. Finally, generator nodes, in case of variation of their own resources with respect to those available to child verifier nodes, may swap their role with one of the child verifier nodes (SW operation), in order to assign to the verification network the most performing nodes.

5 Implementation

The system has been implemented in a single application, named *dcrack*, that is able to perform all the three phases of the attack, *i.e.*, dictionary compilation, passphrase generation and passphrase verification. In such a way, the same application runs on every node of system. The code has been implemented in ANSI C taking into account the requirement of being usable in a multi-platform environment. To this purpose, the application relies only on portable components as shown in Figure 11. In particular, the Apache Portable Runtime (APR) [1] is a set of APIs that guarantees software portability across heterogeneous platforms, through a replacement of functions that are not supported in the underlying operating system. For instance, the use of the APR environment allows to exploit synchronization mechanisms like the "condition-variables", unavailable in the native Windows environment. As to the networking issues, we resorted to the MIDIC middleware, a software layer that provides advanced communication services. MIDIC, in turn, relies on the JXTA technology [8] in order to enable communication between P2P applications, independently of network complexity. For example, if a node accesses to the network through a firewall that enables only HTTP traffic, the middleware automatically establishes a HTTP tunnel in order to guarantee reachability. Both MIDIC and JXTA exploits the APR environment.

The application is subdivided in components, each of which implements a specific function in the node where it runs. The subdivision is made on the basis of a logical classification of activities common to all nodes:

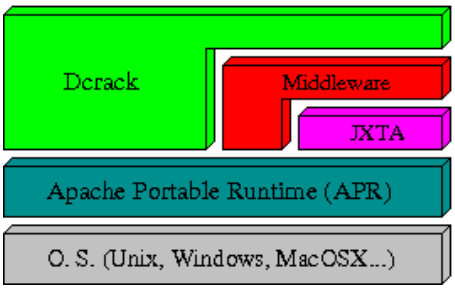


Figure 11. Software structure

execution of task: each task is made of components, the *worker* that acquires and processes information about the attack and *server* that makes available results of required computations;

maintenance operations: such operations are managed by a *controller* component, for what concerns quitting the system and failures, and by a *recruiter* component for the entry of new nodes in the system.

heart-beating activity: this activity is carried out by a *beater* component for sending heart-beating messages to child nodes and by a *Heart* component for receiving such messages.

Active components of the application for the three classes of nodes and communication flows between them are shown in Figure 12. Task messages are sent from the

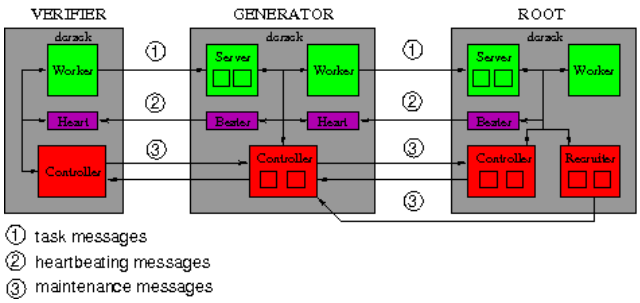


Figure 12. Communication scheme

Work component to the Server component of the application running on the father node. Heart-beating messages are sent from the Beater component of the application running on the father node to the Heart component running on the child nodes. Maintenance messages are exchanged between the controller component of the application running on a child node and the corresponding component in the father node.

Each component runs in a thread whose implementation depends on the platform (but this is transparent to the application since it relies on the APR environment). Cooperation

between threads follows the *work-crew* model. Moreover, threads in charge of components that may require simultaneous communication (*i.e.,* the server, controller and recruiter components) generate a *service* thread to which the communication is demanded. Cooperation between component threads and service threads follows the *boss-worker* model.

6 Experimental Results

We measured the performances of the proposed architecture in a test-bed constituted by a 100baseT Ethernet LAN with 20 personal computers, equipped with a 2.8 Ghz Intel Pentium IV processor and 512Mb of RAM running the Linux operating system. As sample target of the attack, we selected the GnuPG cryptographic software with an ElGamal key having a length of 768 bits.

To generate the dictionary we started from the text of "Divina Commedia" (a famous epic poem of Italian literature) and, as a consequence, generated passphrases are in Italian. In order to evaluate the throughput of the system we chose a passphrase that could not be found with this dictionary, forcing the system to generate and test all passphrases that could be derived from the input text and the defined passphrase generation rules.

Before starting the full experiment, we carried out some preliminary tests, in order to find out how many verifier nodes could be fed by a single generator node. Therefore, the following parameters have been evaluated: k, the number of passphrases that can be checked by a verifier node in a second; t_g, the time required to a generator node to generate k passphrases; t_s, the time required to a generator node to compress and send k passphrases to a verifier node; Our tests showed that a verifier node is able to check about 1000 passphrases per second. A generator node requires 0.6 ms to generate 1000 passphrase and about 10ms to compress and send them. Thus, a generator node needs about 11ms to set up the workload that a verifier node carries out in one second. As a consequence, the adequate ratio R between the number of generator nodes and verifier nodes is given by:

$$R = 1/(t_g + t_s) = (1/0,011) \sim 90$$

In other words, with these settings, each generator node could feed up to 90 verifier nodes.

In the test environment, we used a variable number of nodes but, since the time required to generate, compress and send passphrases is about two orders of magnitude smaller than the time used for verification, in all tests, we used a single generator task that coexisted with the root task on a single node (same computer) of the network.

The results we obtained are very encouraging, since the throughput of the system (measured as the inverse of the time required to test all possible passphrases) increases in a linear way with respect to the number of verifier nodes. We compared these results with those produced by a commercial solution for Grid computing (AGA by Avanade) and found that the throughput of our solution is (about) 20% higher (obviously with the same number of nodes). Finally, no appreciable difference has been found with respect to the operating system of the nodes (PCs we used were dual-boot, so we could test Linux and Windows on the same hardware). As to the reliability of the platform, we checked it in a separate set of tests in which we used up to three generator nodes. Failures of both verifier and generator nodes were successfully managed by the infrastructure by following the procedures described in section 4.3.

7 Conclusions

We presented an architecture to perform distributed dictionary attacks. The system has been tested on a *private keyring* of the GnuPG cryptosystem after a careful study of the features of the encryption system. In particular we devised a technique to quickly check candidate passphrases by limiting the execution of the most expensive control to a subset of the passphrases selected according to much less expensive controls. There are a number of possible directions for future activities. For instance taking into account the results reported in section 6, it is possible to introduce new generation rules and increase their complexity. Besides that, since there are many applications with features similar to cryptographic challenges, another interesting possibility could be to employ the distributed architecture for other computational tasks.

References

[1] Apache portable runtime (apr). http://apr.apache.org/.

[2] The gnu privacy guard (gnupg). http://www.gnupg.org/.

[3] Medussa. http://www.bastard.net/∼kos/medussa/medussa.html.

[4] Rfc2144: The cast-128 encryption algorithm. http://www.faqs.org/rfcs/rfc2144.html.

[5] Rfc2440: Openpgp message formats. http://www.faqs.org/rfcs/rfc2440.html.

[6] Rfc3174: Us secure hash algorithm 1 (sha1). http://www.faqs.org/rfcs/rfc3174.html.

[7] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[8] L. Gong. Jxta: a network programming environment. *Internet Computing, IEEE*, 5(3):88–95, May/June 2001.

[9] T. Perrine and D. Kowatch. Teracrack: Password cracking using teraflop and petabyte resources. *San Diego Supercomputer Center Security Group Technical Report*, 2003.

TPMC: A Model Checker For Time–Sensitive Security Protocols

Massimo Benerecetti Nicola Cuomo Adriano Peron
Dept. of Physical Sciences Dept. of Mathematics Dept. of Physical Sciences
Università di Napoli "Federico II", Napoli, Italy
bene@na.infn.it ncuomo@na.infn.it peron@na.infn.it

Abstract— **In this paper we face the problem of verifying security protocols where temporal aspects explicitly appear in the description. In previous work, we proposed *Timed HLPSL*, an extension of the specification language HLPSL (originally developed in the Avispa Project), where quantitative temporal aspects of security protocols can be specified. In this work we present a model checking tool for the analysis of security protocols which employs THLPSL as a specification language and UPPAAL as the model checking engine. To illustrate how our framework applies, we also provide a specification of the Wide Mouthed Frog protocol and show some experimental results on a number of security protocols.**

I. Introduction

Much work has been devoted to formal specification and analysis of cryptographic protocols, leading to a number of different approaches and encouraging results (e.g. see [18]). Most of the proposed protocol specification languages and verification techniques are limited to cryptographic protocols where quantitative temporal information is not crucial (e.g. delay, timeout, timed disclosure or expiration of information do not affect the correctness of the protocol), and details about some low level timing aspects of the protocol are abstracted away (e.g. timestamps, duration of channel delivery etc). In this context, the specification language HLPSL has been proposed within the Avispa Project (see [1]), for the specification of industrial-strength security protocols. HLPSL allows for modular specifications, specification of control flow patterns, data-structures, and security properties. It is also sufficiently high-level to be used by protocol engineers.

In this paper we focus on the problem of specifying and verifying security protocols where temporal aspects directly affect the correctness of the protocol, and, therefore, need to be explicitly considered both in the specification and the verification. Examples of time sensitive protocols are, for instance, the non-repudiation Zhou-Gollmann protocol [19], the TESLA authentication protocol [15] and the well known Wide Mouthed Frog protocol [4].

The formal framework generally employed to model temporal features, in the context of finite state machines, is that of Timed Automata [3], and the corresponding *model checking* verification techniques are supported by a variety of tools. The model checker KRONOS [9], developed at VERIMAG, supports model checking of branching time requirements. The UPPAAL toolkit [17] allows for checking safety and bounded liveness properties. However, Timed Automata cannot be employed by protocol designer as a specification formalism in itself, being a rather low level formalism, lacking the ability of expressing parallelism and synchronization on structured messages built over cryptographic primitives. In a previous paper [5] we proposed Timed HLPSL (THLPSL for short), as temporal extension of the specification language HLPSL. The temporal feature introduced in THLPSL are: (a) temporal constraints of the control flow (the usual delays and timeouts associated with performing a transition) with respect to the occurrence of some event, (b) duration of a transition, (c) temporal constraints on the availability and usability of messages (message disclosure and expiration time) with respect to the occurrence of some event, and (d) delay in channel delivery. The semantics of THLPSL is formally defined in [5] by a mapping onto eXtended Timed Automata [7] (XTAs for short), the variant of timed automata employed by UPPAAL which, differently from the basic model of Timed Automata, allows for an explicit representation of concurrency and communication in the form of handshaking.

In this paper we describe the TPMC (Timed Protocols Model Checker) tool we developed for the analysis of timed security protocols. TPMC employs THLPSL as a specification language and UPPAAL as the model checking engine. The analysis of a protocol in TPMC consists in a translation of its THLPSL specification into the input language of UPPAAL according to the semantics presented in [5]. For the sake of ease of definition, such a semantics maps THLPSL specification onto pure XTAs, without exploiting the full expressive power of the UPPAAL language, which allows for shared integers variables, and integer and boolean arrays. The use of this additional features allows for exponentially more succinct UPPAAL specifications. As a consequence,

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

the mapping implemented in TPMC is not the one described in the formal semantics, but a semantically equivalent one which, taking advantage of the full expressivity of UPPAAL XTAs, can be more efficiently employed for implementation purposes. We also provide an experimental evaluation of the tool on a number of security protocols.

The idea of using Timed Automata for specifying real time systems and proving security properties is not new (e.g., see [6], [10], [12], [16]). Our approach differs from [6], [12] in that Timed Automata are not the specification language itself, but the back-end of an high level specification language.

The rest of the paper is organized as follows: in the next section we informally present the specification language THLPSL; in Section III we provide the specification the Wide-Mouthed protocol in THLPSL. In Section IV we describe the translation of THLPSL specifications into UPPAAL XTAs. Finally we conclude giving some experimental evaluation of our tool in Section V.

II. Timed HLPSL

In this section we informally describe the main features of the specification language THLPSL, a timed extension of the specification language HLPSL [1] and give an intuition of its semantics. A formal definition of the THLPSL syntax and the semantics, which is given in terms of XTAs [7], can be found in [5].

A strong limitation of the original HLPSL is that it does not allow for explicit specification of temporal aspects such as delays, timeouts, timing constraints on the validity of messages, etc., therefore making it unsuited to specify protocols where temporal aspects may affect correctness. THLPSL extends HLPSL by allowing for expressing the following temporal aspects:
a) temporal constraints on the control flow of participants to a protocol session;
b) duration of a transition, expressed as lower and upper bounds on its duration;
c) temporal constraints on the availability and usability of messages (message disclosure and expiration time);
d) duration of channel delivery, expressed as lower and upper bounds on the channel delay.

THLPSL allows for structured definitions of protocols. A protocol specification consists in the description of a set of roles. It is possible to distinguish between two kinds of roles: *basic roles* which describe the behavior of a participant to a protocol; *composition roles* that compose in parallel instances of basic roles (one for each participant to the protocol sessions) instantiating their parameters with constants.

Roles can be parameterized (with the obvious exception of the main role) and can exploit local variables defined within a local declaration section. Declared variables can be initialized by means an *initialization predicate*. Local variables, formal parameters and constants are typed. THLPSL supports various kinds of types. Common built-in types are the following: `agent` type (for agent names); `channel` type (for communication channel names); `public_key` and `symmetric_key` type (for public and symmetric keys used by cryptographic primitives); `text` type (for text messages); `nat` type (for natural numbers); `function` type (for hash functions). Type `text` may have additional attributes enclosed within brackets. For instance, the type `text(fresh)` is the type for freshly generated nonces.

The type `channel` may have additional attributes enclosed within brackets, e.g., `C:channel(dy,lb,ub)` specifies a channel controlled by a Dolev-Yao (DY) intruder [8] with minimum transmission delay `lb` (a rational number in $\mathcal{Q}_{\geq\prime}$) and maximum transmission delay `up` (a rational number in $\mathcal{Q}_{\geq\prime} \cup \{\infty\}$);

THLSPL provides some form of flexibility in the specification of the constraints on delay/timeout and message disclosure/expiration, by allowing to express these constraints with respect to the occurrence of a transition executed by a participant in the protocol. To this purpose, THLPSL is equipped with a variable type `role_instance` for role instances, which can only be used for formal parameters of roles (and not for the declaration of local variables). Intuitively, a formal parameter `RI` of type `role_instance` will be instantiated with a number between 1 and n in the definition of the main composition role, where exactly n roles are composed in parallel. Therefore, if `RI` is instantiated with number i, then it refers to the i-th role instance in the parallel composition. This allows for expressing time constraints relative to occurrences of events (referred to by transition labels) taking place within specific role instances.

Participants to a protocol session communicate by sending and receiving structured messages. Structured messages are represented by *message terms* which are inductively composed from variables and constants in the following way:
- Variables and constants are terms;
- `X[dt,et,RI,lab]` is a term (timed term), where `RI` is a formal parameter of type `role_instance`, X a variable of type text, text (fresh) or key, `dt` is a rational number in $\mathcal{Q}_{\geq\prime}$ and `ut` a rational number in $\mathcal{Q}_{\geq\prime} \cup \{\infty\}$ and `lab` is a transition label.
- `V'` is a term, with `V` any variable (*priming*);
- `{T}_K` is a term, with `K` variable of type symmetric_key public_key and `T` a term (*encryption*);
- `T1.T2` is a term, with `T1`, `T2` terms (*pairing*);
- `H(T)` is a term, with `T` a term and `H` a variable of type function (*hashing*);

• `inv(K)` and `{T}_inv(K)` are terms, with `K` a variable of type public key (*private key* and *signature*). Priming of variables is used for variable assignments, to refer to the values of the variables after the assignment. A term of the form `X[dt,et,RI,lab]` represents a term `X` that will be disclosed between time `dt` and `et` relative to the execution of the transition labeled `lab` within role instance `RI`, and it will expire after the temporal bound `et`. Moreover, we add a predicate of the form `EXP(X)`, with `X` a variable of type text, text (fresh) or key, which intuitively holds true if `X` is assigned to a message which has expired. We also assume an additional label `start` which represents a fictitious transition taken at time 0 to initialize the main role.

A protocol run start from a, chosen, composition role called *main role*. The behavior of a basic role is described by means of a state-transition formalism. Intuitively, a state of the role instance is determined by the content of its local variables and the value of its actual parameters. Set of states of the role are declaratively denoted by standard boolean expressions over the value of variables and primed variables (e.g. a Boolean expression represents the set of role states where the Boolean expression holds true). A transition allows to leave a state of the role receiving a message (from another participant to the protocol) and to reach another state delivering a message. Transitions are declared in a role by a sequence of transition schemas.

A *timed transition* schema has the form:

```
lab. Pred Rec_Op >>(t1,t2,lb,ub,RI,lab1)
                  Primed_Pred /\ Send_Op
```

where,

• `lab` is a *label* identifying the current transition in the protocol schema;

• `Pred` is the *triggering predicate* defining the set of states from which the transition take place. It is a conjunction of a state predicate `SPred`, and possibly a message predicate `MPred`. `SPred` has the form $\bigwedge_{i=1}^{k} X_i = c_i$, where X_i is a state variable, namely a variable not occurring in any message term in the role, and c_i is a constant. `MPred` is a conjunction of (negations) of atoms either of the form $X = Y$ or $EXP(X)$, where X and Y are message variables occurring in message terms within the role.

• `Rec_Op` is an *optional* receive operation on a channel of the form `C(T)`, where `C` is a channel variable and `T` a term;

• `Primed_Pred` specifies the resulting state of the transition and has the form $\bigwedge_{i=1}^{z} X'_i = c_i$, where X'_i is a primed variable not occurring in any message term of the role and c_i is a constant. In the resulting state, the variables occurring in `Primed_Pred` are assigned the current value of the corresponding constant, and the remaining variables keep their current value;

• `Send_Op` is an *optional* send operation on a channel of the form `C(T)`, where `C` is a channel variable and `T` a term. Possible primed variables in `T` are assigned newly generated values (e.g., fresh nonces generation);

• `RI` is a formal parameters of the current role of type `role_instance`, `t1` and `lb` are rational numbers in $\mathcal{Q}_{\geq 0}$, `t2` and `ub` rational numbers in $\mathcal{Q}_{\geq 0} \cup \{\infty\}$, and `lab1` is a transition label. These parameters specify a transition that will be enabled between time `t1` and `t2` relative to the execution of the transition labeled `lab1` within the role of role instance `RI`, that will complete between time `lb` and `ub`.

THLPSL also allows for untimed transitions. For instance, a transition without any temporal constraints (neither delay/time out nor duration constraints) can be specified by `>>(0,`∞`,0,`∞`,RI,start)`.

An *urgent transition* schema has the form:

```
lab. Pred ->(t1,t2,RI,lab1) Primed_Pred
```

where all the parameters have the same interpretation as in the previous case. The intuitive semantics of an urgent transition is a transition which is enabled between time `t1` and `t2`, relative to the execution of the transition labeled `lab1` within the role instance `RI`, and is forced to trigger as soon as enabled, i.e. without any further delay. Notice that triggering of an urgent transition does not depend upon synchronization with other roles, as it cannot send or receive messages. Urgent transitions are intended to model activities local to a role, which have no duration and must not affect the overall timing.

The composition section of a composition role is used to instantiate other basic and composition roles using the following syntax:

```
R1(actual_parms) /\ R2(actual_parms) /\ ...
```

The intended meaning is that composed roles "run" in parallel with interleaving semantics.

III. Specification of the Wide Mouthed Frog protocol in THLPSL

In this section we consider the well known Wide Mouthed Frog authentication protocol [4]. The protocol involves three participants: Alice, Bob and the Server. Alice sends a message to the Server containing the identity of Bob (the intended receiver), a fresh session key K_{ab}, and a timestamp T_A, encrypted with a symmetric key K_{AS}, shared by Alice and the Server. The Server then checks if the timestamp is recent and, if this is the case, forwards the session key and a new timestamp T_B to Bob, encrypted with a symmetric key K_{BS}, shared by Bob

and the Server. Bob can now check if the timestamp T_S is recent and, if this is the case, accepts the session key as valid. Following is a description of the protocol steps:

1 $A \rightarrow S : A, \{B, K_{ab}, T_A\}_{K_{AS}}$
2 $S \rightarrow B : \{A, K_{ab}, T_S\}_{K_{BS}}$

The idea is that the participants use the timestamps to assess validity of the session key. A session key should be considered valid if the associated timestamp is recent enough. The protocol is known to be vulnerable to reply attacks, where an intruder simply repeatedly intercepts the message sent by the Server and, exploiting the structural similarity of the encrypted components in the two messages, repeatedly replies it back to the Server, who interprets it as a request to establish a new session key between the participants. If the intruder replies are fast enough, it can succeed in forcing the Server to keep the timestamps updated indefinitely, causing a, possibly compromised, session key to be associated to a fresh timestamp.

In order to model the validity of timestamps and session keys in THLPSL, we associate to each of them an expiration time. In particular, the initiator assigns an expiration time to the session key, wide enough to cover the estimated maximum delays of both the communication channels from Alice to the Server and from the Server to Bob. Similarly, Alice (resp., the Server) assigns the expiration time to each generated timestamp. An attack would be detected if Bob receives an expired session key associated with a non expired timestamp.

Below is a possible specification of the protocol, where we assume a maximum delay 5 to the channels connecting the participants. The expiration of the session key is set to the sum of the channels delays. The role for agent Alice is specified as follows:

```
role Alice(A,B,S:agent, SND:channel(dy,0,5),
           Kas:symmetric_key, AI:role_instance)
   played_by A def= :
   local Stat:nat, Ta:text(fresh), Kab:symmetric_key
   init Stat=0
   transition
   a0. Stat=0 >>(0,∞,0,0,AI,start) Stat'=1 /\
       SND(A.{Ta'[0,5,AI,a0].B.Kab[0,10,AI,a0]}_Kas)
end role
```

Notice that the role Alice is parametrized with respect to three agent names (`A, B,S`), one dy channel `SND`, one symmetric key `Kas`, and one role instance parameter `AI`. The `played_by` keyword states that the agent playing the role corresponds to the first agent parameter `A`. In the local variable declaration section the variable `Stat`, of type natural number, a fresh nonce variable `Ta` and a symmetric key `Kab` are declared. The `init` clause opens the variable initialization section, while the `transition` clause opens the section containing transition schemas. The transition schema, labeled `a0`, is a send timed transition which takes from a state where variable `Stat` is equal to 0 to a state where `Stat` is equal to 1, and all the remaining variables, except `Ta`, remain unchanged. The additional effect of the transition is that the term `A.{Ta'[0,5,AI,a0].B.Kab[0,10,AI,a0]}_Kas` is sent over the channel `SND`, where `Ta'[0,5,AI,a0]` represents a fresh timestamp generated and assigned to `Ta` by the transition, with disclosure/expiration interval between time 0 and 5 relative to the execution of the transition `a0` of the current role instance `AI`.

The role for agent Bob is specified as follows:

```
role Bob(A,B,S:agent, RCV:channel(dy,0,5),
         Kbs:symmetric_key, BI:role_instance)
 played_by B def=
 local Stat, Valid:nat, Ts:text, Kab:symmetric_key
 init Stat=0
 transition
 b0. Stat=0 /\ RCV({Ts'.A.Kab'}_Kbs)
           >>(0,∞,0,0,BI,start) Stat'=1
 b1. Stat=1 /\ not EXP(Ts) /\ not EXP(Kab)
  ->(0,∞,BI,start)
     Stat'=2  /\ Valid'=1
 b2. Stat=1 /\ not EXP(Ts) /\ EXP(Kab)
  ->(0,∞,BI,start)
     Stat'=2 /\ Valid'=0
end role
```

As to Bob's role, the first transition is a receive transition which requires that another party synchronously sends a message along the channel `RVC`, and that the sent message conforms to the structure of the term `{Ts'.A.Kab'}_Kbs`. The primed variables `Ts'` and `Kab'` in the received term are assigned, after the transition is executed, the value of the corresponding subterm in the unifying received message. The last two transitions are urgent transitions (always enabled) which test the validity of the timestamp and of the key and accept (resp., reject) the key by assigning the value 1 (resp., 0) to the boolean variable `Valid`.

The Server role is specified as follows:

```
role Server(A,B,S:agent, RCV,SND:channel(dy,0,5),
         Kas,Kbs:symmetric_key, SI:role_instance)
 played_by S def=
  local Stat:nat, Ts:text(fresh),
   Ta:text, Kab:symmetric_key
  init Stat=0
  transition
  s00. Stat=0 /\ RCV(A.{Ta'.B.Kab'}_Kas)
       >>(0,∞,0,0,SI,start) Stat'=1
  s01. Stat=1 /\ not EXP(Ta) >>(0,∞,0,0,SI,start)
       Stat'=3 /\ SND({Ts'[0,5,SI,s02].A.Kab}_Kbs)
end role
```

Notice that both the Server and Bob check for non expiration of timestamps (`not EXP(Ta)` and `not EXP(Ts)`) before proceeding (resp., before accepting the session key). Moreover, the Server sets expiration of the timestamps it generates relative to the transition generating it. To model possible acceptance by Bob of an invalid key, we use a variable `Valid` in

Bob's role, which is set to 0 (transition `b2`) if the accepted key has already expired, and to 1 (transition `b1`), otherwise.

The main role `Main` instantiates one instance of role Alice, one of the role Bob and three of the role Server. Roles are instantiated by associating actual parameters (i.e., constants) to formal ones. The resulting role instances are composed in parallel.

```
role Main()
 def=
 composition
   Alice(A,B,S,Snda,Kas,0) /\ Bob(A,B,S,Rcvb,Kbs,1)
   /\ Server(A,B,S,Snda,,Rcvb,Kas,Kbs,2)
   /\ Server(B,A,S,Sndb,Rcva,Kbs,Kas,3)
   /\ Server(A,B,S,Snda,Rcvb,Kas,Kbs,4)
end role
```

A simple property requiring acceptance only for valid keys is the following CTL formula $AG\neg(Alice0.Stat = 1 \wedge Bob1.Stat = 2 \wedge \neg Bob1.Valid)$, which can be checked by the model checker UPPAAL. The property is false, as it is possible for role instance `bob` to accept as valid a key after it has expired.

IV. From THLPSL specifications to UPPAAL XTAs

In this section we shall show how to encode a $THLPSL$ specification into a XTA suitable for model checking in UPPAAL.

UPPAAL XTA are an extension of Timed Automata. A Timed Automaton TA is a finite state automaton enriched with a set of real-valued clocks whose value can constrain the triggering of transition (for a formal definition and a account of the semantics see [3]). Transitions of a TA have the form $\langle l, \phi, a, \lambda, l' \rangle \in \delta$, written also $l \stackrel{\langle \phi, a, \lambda \rangle}{\longrightarrow} l'$, which represents a transition from the location l to the location l' on input symbol a; the *guard* ϕ is a constraint on clocks, and specifies when the transition is enabled; and the *update* set $\lambda \subseteq CK$ states the set of clocks to be reset on executing the transition. An eXtended Time Automaton[7] is the parallel composition $A_1 \parallel \ldots \parallel A_n$ of a collection of Timed Automata $A_1, \ldots, A_n$, in the style of CCS [14]. Automata communicate by means of channels and the communication style is handshaking. Input symbols of TA are replaced by channel names in XTA. If a is the name of a communication channel, then the symbol $a?$ denotes the receiving action over channel a, while the symbol $a!$ denotes the sending action over channel a. In addition, XTA can use (boolean and integer) variables and arrays. Therefore, the guard ϕ of a XTA transition may also constraints values of variables and array elements besides clocks. The update λ is generalized allowing also assignments involving variables and arrays.

As previously said, the formal semantics of $THLPSL$ has be given in [5] by translation into a network of timed automata. In such a translation a timed automaton is provided for each instance role and a timed automaton is provided for the intruder. States of both the participants and the intruder are structured and, in particular, encode besides control information also the knowledge of the playing part at the represented stage of the interaction. Knowledge is suitably encoded by sets of ground instances of message term (ground messages).

The intruder's knowledge is a monotonically increasing set of structured messages. A DY intruder can send to role instances any structured message that it can derive from its knowledge. For every received message the intruder can extract any possible submessage, compatibly with its knowledge of the necessary cryptographic keys. Conversely, known submessages can be recombined freely, using the algebra of message operators. Since structured message provide an unbounded use of pairing cryptographic encoding operators, the number of message the intruder can possibly build is unbounded. However, the messages composed by the intruder which are relevant for the protocol are those unifiable with the message patterns expected by the role instances. Even considering a bounded set of messages, the fact that the intruder can compose and/or modify communicated messages results in an explosion in the number of states (which depend on the subset construction of the set of received messages) and in the number of transitions. For a succinct encoding, the translation implemented in TPMC exploits the ability of UPPAL of handling XTA specifications enriched with variables and arrays. Arrays and variables are used both to encode the knowledge of the role instances and of the intruder, as well as the intruder's ability to compose and decompose messages. In particular, the intruder's knowledge is encoded by a boolean array `K`, where each location represents either a structured message sent along a channel, or a (sub)message obtained by composition/decomposition of known messages. A location of the array `K` is set to `true` when the intruder knows the corresponding (sub)message. Similarly, each role instance `ri` is encoded by an array of integers $\mathtt{N}_{ri}$, which contains the current ground instance associated to each variable occurring in a send or receive operation of the corresponding role.

Communication between role instances is not direct, but implemented by a pair of synchronizations, one between the sender and the intruder and one between the intruder and the receiver. Since communication in the formalism of XTA takes the form of pure communication, a different channel is provided for each conveyed message. Therefore, for each pair $< M, CHN >$, with m a structured ground message sent (resp., received) by a role instance and

CHN a channel name (intuitively, the channel where the message has been sent), a XTA syncronizzation channel named `C_CHN_s_m` (resp., `C_CHN_r_m`) is created.

Since delay/timeouts of timed transition and disclosure/expiration of timed messages are specified relative to a transition label, in order to model these timed feature a clock named `CK_lab_ri` is associated to every pair ⟨`lab`,`ri`⟩, such that transition label `lab` and role instance `ri` occur among the parameters of some timed transition or timed message term. Moreover, a boolean array `F` is used to record, for each transition label referenced within a timed message term or timed transition, whether it has been already executed. An additional clock named `CK_start` is used to model timed constraints referencing the special label `start`, corresponding to the initialization time of the main role. To model the duration of transitions taken by role instances, a local clock named $\mathtt{d}_{ri}$ is associated to role instance `ri`. To model channels delays, to every channel CHN, for which a delay constraint is specified, a clock `CK_CHN` is added. Finally, in order to model disclosure/expiration of timed messages, two boolean arrays `D` and `E` are used, which record whether a timed message has been disclosed or has expired, respectively.

The translation of a $THLPSL$ specification generates:
- an automaton for each role instance;
- an automaton for the intruder;
- an automaton (the *Time Machine*) responsible for handling disclosure and expiration of timed messages.

As to the generation of the automata for the role instances, the first step consists in collecting, by means of a fixpoint construction, the set GM of ground messages and the set $TM \subseteq GM$ of timed messages possibly generated by the protocol participants and the intruder, according to a typed model. This phase defines, for each role instance `ri`, the following correspondences, recorded in suitable data structures:
i. a function $\rho_{ri} : MVar_{ri} \longrightarrow 2^{GM}$ mapping each message variable of the role of `ri` onto a set of possible instances of that variable;
ii. for every message term M occurring in a receive operation in the role of `ri`, a function $\chi^M_{ri} : Var'_{ri}(M) \longrightarrow 2^{GM}$, mapping the primed variable occurring in M ($Var'_{ri}(M)$) onto sets of possible instances of ground messages.
The function ρ_{ri} encodes a set of possible evaluations for the message variables of the role instance, in the sense that for a message variable X, $\rho_{ri}(X)$ gives the set of possible values of the vector element $N_{ri}[X]$, up to a suitable encoding of ground messages into integers. Function χ^M_{ri} encodes the result of a structure preserving unification mechanism between message terms expected by the receiver and ground terms sent by a sender (see [5] for a formal account).

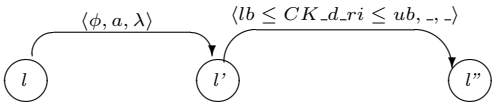


Fig. 1. XTA transitions encoding a timed transition.

In the following we sketch the construction of the instance role automaton for `ri`. For the sake of presentation, some of the technical details are omitted. Each location of a role instance automaton represents a location in which some state predicate holds. Let L be the set of atoms of the form $X = c$, such that $X = c$ occurs in a transition `SPred` or of the form $X' = c$, such that $X' = c$ occurs in a transition `Primed_Pred`. The set of locations of the role instance automaton for `ri` are in correspondence with subsets of L.

The general form of a sending timed transition is:

```
lab. SPred /\ MPred >>(t1,t2,lb,ub,RI,lab1)
                      Primed_Pred /\ CHM(M)
```

Each $THLPSL$ transition defines a set of pairs of XTA transitions, a pair for each possible instantiation (given by the functions ρ_{ri} and χ^M_{ri}) of the message variables $\{X_1, ..., X_k\}$ occurring in the transition, as shown in Fig. 1. The first XTA transition models the effect of the THLPSL transition, while the second one models its duration. With reference to Fig. 1, given $\theta = \{m_1, ..., m_k\}$ a possible instantiation of the message variable $\{X_1, ..., X_k\}$ according to ρ_{ri}:
- l is a location corresponding to a set of atoms in L which contains all the atoms in `SPred`.
- l' is a location corresponding to a set of atoms in L which contains all the atoms of the form $X = c$, such that $X' = c$ occurs in `Primed_Pred`, and all the atoms $X = c$ occurring in `SPred` such that X' does not occurs in `Primed_Pred`.
- l'' is a distinct copy of l', introduced to model transition duration.
- ϕ is a conjunction of: (i) atoms of the form $N_{ri}[X_i] = m_i$ for every message variable X_i occurring unprimed in the message term M; (ii) atoms of the form $N_{ri}[X_i] = N_{ri}[X_j]$ (resp., `not` $N_{ri}[X_i] = N_{ri}[X_j]$), for every atom of the form $X_i = X_j$ (resp., `not` $X_i = X_j$) occurring in `MPred`, and atoms of the form $E[X_i] = 1$ (resp., $E[X_i] = 0$), for every atom of the form $EXP(X_i)$ (resp., `not` $EXP(X_i)$) occurring in `MPred`; and (iii) the clock condition $t_1 \leq$ `CK_lab1_ri` $\leq t_2$.
- a is `C_CHM_s_m!`, where m is the ground message obtained by substituting $\{X_1, ..., X_k\}$ by θ.
- λ is a set of assignments contag `F[lab_ri]` := 1, `CK_d_ri` := 0, `CK_lab_ri` := 0, $N_{ri}[X_i] := m_i$ for each X_i occurring primed in M.

Notice that the transition guard ϕ enables the transition when the current state: (i) assigns to the un-

primed variables the ground messages assigned by θ; and (ii) satisfies all the conjuncts in `MPred`. The update λ sets the flag `F[lab_ri]` to record the execution of the transition, resets the clocks associated to the transition, and assigns fresh values to the primed variables in the message term sent. The general form of a receive timed transition is:

```
lab. SPred /\ MPred  /\ CHM(M)
     >>(t1,t2,lb,ub,RI,lab1) Primed_Pred
```

Each receive $THLPSL$ transitions defines a set of XTA transitions, one for each possible instantiation (given by the functions ρ_{ri} and χ^M_{ri}) of the message variables $\{X_1, ..., X_k\}$ occurring in the transition. Given $\theta = \{m_1, ..., m_k\}$ a possible instantiation of the message variable $\{X_1, ..., X_k\}$ according to ρ_{ri}, and $\psi = \{u_1, ..., u_z\}$ a possible matching, according to χ^M_{ri}, for the message variable $\{Y_1, ..., Y_z\}$ occurring primed in M, a pair of XTA transitions as in Fig. 1 is added, where:

- l, l', l'' and ϕ are defined as for send transitions;
- a is `C_CHM_r_m?` where m is the ground term obtained by substituting the message variables in $\{X_1, ..., X_k\}$ which occur unprimed in M by θ and all the message variables in $\{Y_1, ..., Y_z\}$ by ψ;
- λ is a set of assignments containing `F[lab_ri]` := 1, `CK_d_ri` := 0, `CK_lab_ri` := 0, $N_{ri}[Y_i] := u_i$, for each $Y_i \in \{Y_1, ..., Y_z\}$.

To guarantee that the duration of a timed (send or receive) transition is modeled correctly, the intermediate location in Fig. 1 is equipped with the invariant[1] $lb \leq CK\_d\_ri \leq ub$.

Since urgent transitions cannot send or receive messages, neither synchronization nor update is necessary. Therefore, they are encoded as XTA transitions between a starting location l to an ending location l' defined as in the previous cases, with the addition that the starting location is set urgent, and the guard condition is a conjunction of atoms of the same form as those defined for cases (ii) and (iii) for timed send or receive transitions.

To model a DY intruder, the intruder automaton plays the role of the communication channel between the role instances, and it is allowed to compose, decompose, forward, block and delay messages. The automaton has a single location and loop transitions for sending known messages to role instances, receiving messages sent by role instances and composing/decomposing messages.

For every ground message $m \in GM$ and channel CHN, there is a loop transition for a send action, whose decoration $\langle \phi, a, \lambda \rangle$ is
$\langle$K[m] = 1 $\wedge$ `CK_CHN` $\geq lb$, `C_CHN_r_m!`, _$\rangle$,

[1]Invariants are associated to locations; remaining in a location is allowed as long as the invariant holds true.

where `CK_CHN` is the clock associated to channel CHN to model channel delay.

For every a ground message $m \in GM$ and channel CHN, there is a loop transition for a receive action, whose decoration $\langle \phi, a, \lambda \rangle$ is
$\langle$_, `C_CHN_s_m?`, `K[m] := 1; CK_CHN := 0`$\rangle$

Transitions for composition/decomposition of messages encode the standard rules of a DY intruder. For instance, if the intruder knows two ground messages m_1 and m_2 and $m_1.m_2 \in GM$, then it also knows $m_1.m_2$ (and vice versa). Similarly, if it knows a ground messages $\{m_1\}_k$ and a ground key k, and $m_1 \in GM$, then it also knows m_1 (and vice versa). The loop transitions for the above two composition, decomposition actions have the following decorations: the former is $\langle K[m_1] \wedge K[m_2], \_, K[m_1.m_2] := 1\rangle$ ($\langle K[m_1.m_2], \_, K[m_1] := 1; K[m_2] := 1\rangle$), and the latter is $\langle K[\{m_1\}_k] \wedge K[k], \_, K[m_1] := 1\rangle$ ($\langle K[m_1] \wedge K[k], \_, K[\{m_1\}_k] := 1\rangle$).

The *Time Machine* automaton (TM) is responsible for handling disclosure and expiration of timed messages by updating the boolean arrays D and E. The array F is used to record the execution of a transition referenced by some timed message (or timed transition). Therefore, disclosure or expiration of a timed message relative to a given transition is performed only if the referenced transition of role instance `ri` labeled `lab` has been executed (`F[lab_ri]` = 1).

For every ground message m which is an instance of a timed variable message `X[dt,et,RI,lab]` in role instance `ri`, a loop transition for disclosure is added, whose decoration $\langle \phi, a, \lambda \rangle$ is

$$\langle \texttt{F[lab\_ri]} = 1 \wedge \texttt{not D[m]} \wedge \texttt{CK\_lab\_ri = dt}, \_, \texttt{D[m]} := 1 \rangle$$

and a loop transition for expiration is added whose decoration $\langle \phi, a, \lambda \rangle$ is

$$\langle \texttt{F[lab\_ri]} = 1 \wedge \texttt{not E[m]} \wedge \texttt{CK\_lab\_ri} = et, \_, \texttt{E[m] := 1} \rangle$$

To guarantee disclosure/expiration transition at due time without any further delay, the location of TM is equipped with an appropriate invariant. The invariant is a conjunction, over all the ground instances m of the timed message terms of the form `X[dt,et,RI,lab]` within role instance `ri`, of constraints of the form:

$$(\texttt{F[lab\_ri]} \wedge \texttt{not D[m]}) \rightarrow \texttt{CK\_lab\_ri} \leq \texttt{dt}) \wedge$$
$$(\texttt{F[lab\_ri]} \wedge \texttt{D[m]} \wedge \texttt{not E[m]}) \rightarrow \texttt{CK\_lab\_ri} \leq \texttt{et})$$

V. Conclusions

We have implemented a prototype model checker TPMC in C++. The tool integrates a compiler from THLPSL specifications to UPPAAL XTAs with the model checking engine provided by UPPAAL. To assess the efficiency and scalability of the tool, we ran

Protocol	Inst	CT	VT	Max Inst	CT	VT
WMF	1-1-3	.01	.44	2-2-5	.06	337.37
WMF_{Fix}	1-1-3	.01	.03	2-2-5	.06	111.31
NSPK	3	.02	.86	5	.68	14.75
$NSPK_{Fix}$	3	.02	.10	5	.12	26.42
ISO1	1	.07	.31	8	326.79	339.96
PBK	2	.01	.02	8	.38	15.29
PBK_{Fix}	2	.01	.02	8	.30	138.18

TABLE I: Experimental results.

TPMC on a number of timed and untimed protocols. An excerpt of the results of our experiments is given in Table I. The table shows the results of TPMC for the original and fixed version (as proposed by Lowe [13]) of the Wide Mouthed Frog protocol, and the following untimed protocols: the Needam–Schroeder Public Key protocol (both in the original and fixed version), the PBK protocol (both in the original and fixed version), and the ISO1 protocol (the specifications of these protocols are taken from the AVISPA library of protocols [1]). All the tests are parametric in the number of sessions, where a session involves from two to three participants, depending on the protocol analyzed. Clearly, the bigger the number of sessions, the higher the number of agents and of ground messages sent/received, leading to a growth in the state space to be analyzed. The column Inst. of the table reports the number of sessions of the corresponding test except for the Wide Mouthed Frog protocol, where it reports the number of instances of role Alice, of role Bob and of role Server, respectively. The property verified for the Wide Mouther protocol is the one reported in Section III, while the property checked for all the untimed protocols is strong authentication. We only report the results for the minimal and maximal instance of the protocols we tried to analyze with TPMC. The table reports both the time in seconds spent by the compiler (CT) from THLPSL to UPPAAL and the time in seconds spent by UPPAAL (VT) in the verification phase. The experiments have been run on a 3.0GHz Pentium IV with 1Gb of memory running Linux (Slackware 11.0). On all tests, TPMC correctly reports the expected attack on the flown version of the protocol and no attacks for the fixed versions. Notice that, TPMC allows for both the specification of timed and untimed protocols. Even though TPMC is not optimized for handling untimed protocols, the tests show on those protocols performances which are comparable with those of available tools for untimed protocols (see, e.g., [1]).

We are currently working on the specification and verification of the TESLA protocol[15] as well as other time dependent protocols. We also plan to investigate different forms of intruder, e.g., intruders with restricted abilities compared to a DY intruder, or intruders whose actions take non negligible time. We are also working on an extension of language for security goals so as to allow for time dependent goals.

References

[1] AVISPA: Automated Validation of Internet Security Protocols and Applications. `http://avispa-project.org`.

[2] R. Alur, T. Henzinger, M. Vardi, Parametric real-time reasoning, STOC 1993, pp.592-601.

[3] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science, 126, pp. 183-235, 1994.

[4] M.Burrows,M.Abadi,and R.Needham, A logic of authentication, ACM Trans. on Computer Systems, 8(1):18–36, 1990.

[5] M. Benerecetti, N. Cuomo, and A. Peron, Timed HLPSL for specification and verification of time sensitive protocols. Proceedings of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'06), Seattle, August 15–16, 2006.

[6] R. Barbuti, N. De Francesco, A. Santone, L. Tesei, A Notion of Non-Interference for Timed Automata, Fundamente Informaticae, 54(2-3), 177-150, 2003.

[7] J. Bengtsson, W. Yi: Timed Automata: Semantics, Algorithms and Tools. Lectures on Concurrency and Petri Nets 2003: 87-124

[8] D. Dolev, A.C. Yao, On the Security of Public-Key Protocols, IEEE Transactions on Information Theory, 29(2):198–208, 1983.

[9] C. Daws, A. Olivero, S. Tripakis, S. Yovine, The tool KRONOS, In Hybrid Systems III: Verification and Control, LNCS 1066, pp. 208-219, 1996.

[10] R. Gorrieri E. Locatelli, F. Martinelli, A simple Language for Real Time Cryptographic Protocol Analysis, ESOP 2003, LNCS 2618, pp. 114-128, 2003-03-27

[11] Leslie Lamport, The Temporal Logic of Actions, in ACM Transactions on Programming Languages and Systems, Vol. 16(3), ACM Press, pp. 872–923,1994.

[12] R. Lanotte, A. Maggiolo-Schettini, S. Tini, Timed Information Flow for Timed Automata, submitted.

[13] Gavin Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[14] R. Milner. A Calculus for Communicating Systems. LNCS 92, 1980.

[15] A. Perrig, R. Canetti, J. D. Tygar, D. Song, Efficient Authentication and Signing of Multicast Streams over Lossy Channels. IEEE Symposium on Security and Privacy 2000: 56-73.

[16] M. Napoli, M. Parente, and A. Peron. Specification and verification of protocols with time constraints. Electronic Notes in Theoretical Computer Science, 99:205–227, 2004.

[17] K. Larsen, P. Petterson, W. Yi, UPPAAL in a nutshell, Springer International Journal of Software Tools for Technology Transfer, 1, 1997.

[18] C. Meadows, Formal methods for cryptographic protocol analysis: emerging issues and trends. IEEE Journal On Selected Area in Communications, 21, 2003

[19] J. Zhou, D. Gollmann, An Efficient Non-repudiation Protocol, 10-th Computer Security Foundation Workshop (CSFW'97), Rockport, Massachusets, USA, 126–132, 1997.

AN EXPERIENCE-BASED INCIDENT RESPONSE SYSTEM

G. Capuzzi, E. Cardinale, I. Di Pietro, L. Spalazzi
Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione
Universita' Politecnica delle Marche
60100 Ancona
E-mail: {capuzzi,cardinale,dipietro,spalazzi}@diiga.univpm.it

ABSTRACT

This paper presents a tool for attack detection, attack identification and attack response. These activities have received a great attention by the research community and by several organizations (e.g., ISO/IEC and CERT). Nevertheless, most of the work focuses on the detection and the identification of intrusions instead of attack identification and response. In our work, an **intrusion** is a detectable atomic action performed by an attacker against a given target, whereas an **attack** may go through several phases. Each phase involves different methods and different goals. Therefore, according to our meaning, an attack is a sequence of intrusions.

In our approach, the attack identification and response can be fulfilled in four distinct phases. The *first phase* deals with *intrusion detection*. This means collecting data from several sensors on the network and on computers, e.g., log files of operating systems and system servers, firewalls, (network-, host-, application-based) IDSs. The *second phase* deals with *alarm correlation*. This means correlating all the data collected in the previous phase to the end of providing an attack description in terms of sequence of events as complete as possible. The *third phase* deals with *identification* by means of an experience-based model. This means: the tool has a case memory that contains past attacks described in terms of their features and their corresponding event sequences. This allows us to have a tool capable of identifying an attack on the base of its similarity with previous experiences, and learning new kinds of attacks. The *fourth phase* deals with response. We can link to each past attack an appropriate response plan, then, after the attack identification, we can reuse (after an appropriate adaptation) the plan associated to the recognized attack. This allows us to have a tool capable of identifying an attack on the base of its similarity with previous experiences, and eventually learning new kinds of attacks.

INTRODUCTION

What makes a security manager better than another one? The trivial answer is: her/his experience. The greater her/his experience, the greater it is the number of attacks examinated in the past that can be recognized if they occur again. The greater her/his experience, the greater it is the number of past attacks to which the security manager has found an appropriate responses, the greater it is the possibility of finding a response to a new attack. This is the focus of our work. We propose a system that, at first, supports the security manager in all her/his activities related to attack response (detection, identification, reporting and response) and, second, is an experience based system. Let us explain this idea depicting two different scenarios. In the first scenario, let us suppose to have an attack similar to a past attack: our system has to recognize it like a new occurrence of the past attack. If in the past we have successfully responded to this attack with certain actions, it is reasonable to suppose that these actions may be applied to the new attack as well, even if after an appropriate adaptation. Our system has to retrieve the past plan and adapt it to the current attack. This adapted plan is presented to the security manager that decides whether it has to be modified and/or executed. Part of plan actions must be executed by security manager or other human operators; the rest must be executed by computers. Therefore, our system has to overview plan execution, automatically run the part of the plan that that must be executed by computers, and monitor results. In the second scenario, let us suppose to have an attack that never occurred. Even in this case, our system is able to select past attacks from its database (with their response plans) that share some (few) characteristics with the current attack. The security manager can thus build a response plan for the current attack on the basis of these past response plans. The description of the new attack and the related response plan may be retained and thus improving system experience.

The paper is structured as follows. Section provides a system overview and the related work. In Section , we discuss attack detection, in other words: sensing, normalization, and correlation. The case memory is described in Section . Section deals with the entropic filter that aims to reduce the number of falses. Section deals with attack identification. In Section , we describe the structure of the response plan and the adaptation process. Some Conclusions are drawn in Section .

SYSTEM OVERVIEW AND RELATED WORK

A network and computer *security incident* is "any adverse event whereby some aspect of computer secu-

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

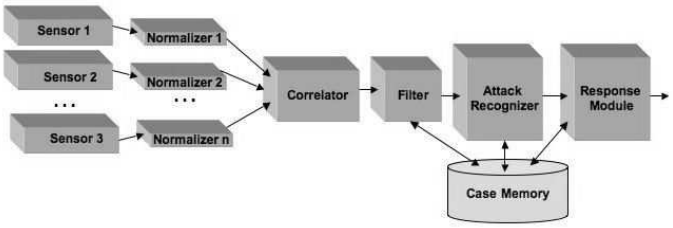


Fig. 1. The architecture of an experience-based system for attack identification

rity could be threatened: loss of data confidentiality, disruption of data or system integrity, or disruption or denial of availability" [16]. Therefore, the security policy of a given organization should provide appropriate *incident response plans*, as remarked, for instance, by ISO/IEC [11] and by CERT [4]. Incident response can be defined as *the detection of an attack to a computer system, its identification and reporting, and the implementation of appropriate responsive actions until normal conditions have been restored*. According to the so called "Plan-Do-Check-Act" (PDCA) model that is adopted in the international standard ISO/IEC 17799 [11], *detection* and *identification* activities of a response plan are part of the "Check" phase, whereas the *reporting* and *responsive* activities are part of the "Act" phase.

The aim of our work is the definition of an incident response tool based on experience, that must be able to plan and perform all the activities listed above: attack detection, identification, and response. As far as we know, there are no examples of systems that cover all these activities, but systems that only deal with some of them. For example, the tools described in [7] and [8] deal with attack identification and management phases, but not with response.

In our approach, incident response can be fulfilled in three phases: *attack detection*, *attack identification*, and *attack response*. As a consequence, our system has the architecture depicted in Figure 1.

The *first phase*, *attack detection*, deals with *sensing*, *normalization*, *correlation*, and *filtering*. This means collecting data from several *sensors* on the network and on computers, e.g., log files of operating systems and system servers, firewalls, (network-, host-, application-based) IDSs. These data must be normalized by *normalizers* since each log file has its own format. Unfortunately, these tools produce noise as well, hence we need correlation and filtering to reduce the high volume of the log messages. A first kinf of filter is made by normalizers themselves. Indeed, log files of an operating system or a web server contain attack alarms as well as data that do not concern attacks. Furthermore, all the data collected by normalizators must be grouped (correlated) in order to find sequences of events that refer to the same attack. For this reason, data from normalizers are sent to the *correlator* module. The correlator output is a set of attacks, each one represented like sequence of events. Nevertheless, after correlation, we still have false or not significative attacks, and thus we need a further filter. Therefore, the correlated event sequences are passed through an *entropic filter*. For each sequence of correlated events is computed its entropy from the entropy of each event in the sequence. The entropy of an event is a logarithmic function of its frequency. Only the sequences with an entropy greater than a given threshold are considered real attacks. This step helps us to avoid a response for very common and not significative sequences as sequences of portscanning or ping. Intrusion Detection Systems (IDS) detect single events (e.g., see [5], [6], [2], [14]), but they are not able to draw the whole picture of an attack. They are not effective to deal with many correlated intrusions involving multiple entities of a computer and network system over time.

The *second phase* deals with *identification*. In our approach, the identification can be accomplished by means of an experience-based model. This means: the proposed system includes a case memory which contains the experience of the system. In other words, it contains a set of past attacks with their features and the related response plans. Each attack is represented as a set of features (attack description, priority, and so on) and a sequence of alerts. For each attack as well as for each event of an attack, the case memory also records the entropy. The entropy of an attack is a function of the entropy of its events. This allows us to have a tool capable of identifying an attack on the base of its similarity with previous experiences, and eventually learning new kinds of attacks. Indeed, when a new sequence that is considered a real attack by the entropic filter is passed to the *attack recognizer*, the tool searches for the most similar past attack (contained in the case memory) to the current one (according to an appropriate similarity metric) and returns the corresponding identification. We use four similarity metrics. Two of them are simple but effective similarity metrics based on pattern-matching. The other ones are based on the entropy of attacks. The attack recognizer module selects the top ranked case with a similarity greater than a given threshold. When no past attacks have been found, the new attack can be retained in order to improve the case base, i.e., the tool experience. As far as we know, we have only few examples of security tools that are able to identify an attack. For instance, [15] proposes a model based on attack profiles; these profiles can be used to identify an on-going attack. Nevertheless, that work does not propose how this can be automatically fulfilled. Moreover, we have some products that work on attack identification, for example [7] and [8]. These tools gather many types of log messages and correlate them, in order to make it easier to analyse them.

The *third phase* deals with *attack response*. In the case memory, we have an appropriate response plan to each past attack. Then, after the attack's identification, we reuse this plan properly adapted. Here the idea is: if the the current attack is similar to the retrieved past attack, then an appropriate response plan for the current attack should be similar to the plan

of the retrieved past attack. We should need only few adaptations. We use a simple adaptation algorithm before executing the plan. This plan is returned to the security manager who checks and corrects it. After its validation, the security manager can allow plan execution. A plan is a sequence of actions, each action can be an action that must be automatically executed by a software, or an action that must be performed by the security manager. After the execution stage, the system stores the plan updating the case memory. In this way, the system is able to learn from new cases and to improve the effectiveness against new attacks. There are tools that work in this phase, but they have a different function. They provide response plans quite different from ours, consisting of generic advices to create a security plan for the network or several advices to avoid insecure behaviours. Our system provides response plans consisting of detailed actions to assess the incident, restore the previous status, and improve the security level of the network. We have just an example where the returned plans are similar to our kind of plans (see [17]). That system is based on classical AI planning. When it receives alerts from IDSs, it establishes what are the goals to satisfy in order to react to the attack. It has a set of actions described in terms of preconditions and affect and combines them in order to reach the goal from the current situation. This solution is flexible when we have never recorder attacks. Nevertheless, classical AI planning has been recognized to be a complex task (usually it is undecidable). It is based on the assumption to have "a complete theory" able to describe all the possible actions we need to use. Unfortunately, this is usually true only for toy examples. In our system , we prefer to trust the experience. Indeed, an experience-based approach is known to be extremely useful in the diagnosis and the management of several different kinds of emergency [10], [3], [9], [13]. For instance, the work of [10] deals with alarm correlation (i.e., with identification), even if not with response. Furthermore, the system deals with fault alarms, not intrusion alarms, and the goal is to obtain a fault tolerant network, not a secure network. The work of [3] and the work of [9] deal with intrusion detection. There the goal is to exploit learning to improve detection and avoid the need of frequently updating the database of known attacks. In our work, the goal is the improvement of attack identification and response by means of learning. We have just an example of application to incident response [13]. There the goal is to improve detection and to avoid the need of frequently update the database of known attack.

ATTACK DETECTION

Attack detection is fulfilled in three steps: *intrusion detection*, *alarm normalization*, *alarm correlation*.

Intrusion detection consists of sensing all the alerts that can provide us information about possible attacks. This mainly relies on sensing data, such as data obtained by monitoring traffic on a network, activity logs stored on a computer, or system state. We use both signature recognition and anomaly detection sensors.

Sensed data include a lot of irrelevant information, and cause difficulty for efficient and accurate attack detection. For example, the log file of an operating system contains a vaste range of log messages, only few of them deal with intrusions, the rest deals with normal system activities or errors. In other words, we have an event log everytime a device is mounted or unmounted, everytime the clock must be synchronized, or everytime a system error occurs. Furthermore, each sensor has its own output data format. Therefore, the collected data must be reduced in number and ordered in a single specific format. For this reason, for each sensor we have developed a module to have a preliminary filtering of sensed data and to normalize them in a specific format. The preliminary filter is composed by a service-based white-list: applications not included in the white-list are not monitored. For example, we do not need to take into account alarms related to FTP when the FTP service is not active on that host. Concerning the normalization, for alarm we consider the following parameters: the tipology of the sensor that produced the alert, the timestamp, the destination port, the source IP, the target IP and the related message. We developed different modules in C++ and Python, one for each sensor, to normalize its output in the specified format. Normalized alerts represent the input of the correlator.

The correlator is the module demanded to correlate alerts coming from different sensors and producing an attack descriptor list, composed of sequences of alerts. It also reduces redundant alerts, by a fusion process. This module is a variant of the correlator proposed by [1]: we only considered the following subsystems:

- fusion: it correlates several instances of the same event detected by different sensors; for instance, a malformed packet sent to the DMZ subnet should be detected by the perimetral sensor and by a sensor on the DMZ
- session reconstruction: it correlates alerts, detected by host intrusion detection and network intrusion detection systems; for instance, an attack launched against a web server should be detected by a NIDS and by a HIDS installed on the web server
- focus recognition: it correlates one to many attacks (for instance, a portscanning), consisting of alerts that have the same source and different targets and many to one attack (for instance a DDoS attack), consisting of alerts that have the same target and different sources
- thread reconstruction: it links all the alerts of the same attack, looking for alerts from the same source to the same target

In addition, we did not consider the IDMEF format for alerts description.

CASE MEMORY

In the case memory, each *case* consists of two components: a static component and a dynamic component. The dynamic component is given by the list of

Step	Intrusion Type	Sensor	Source	Target
1	TCP portscan	NIDS2	207.46.176.50	172.16.113.84:80
2	SNMP trap tcp	NIDS1	207.46.176.50	172.16.113.84:444
3	SNMP trap tcp	NIDS2	207.46.176.50	172.16.113.84:444
4	SNMP AgentX/tcp request	NIDS1	207.46.176.50	172.16.113.84:80
5	SNMP request tcp	NIDS1	207.46.176.50	172.16.113.84:135
6	SCAN nmap XMAS	NIDS1	207.46.176.50	172.16.113.84:80
7	BACKDOOR NetSphere Access	NIDS1	207.46.176.50	172.16.113.84:69

Fig. 2. An example of attack.

ID	Attack Type	Intrusion Type	Source	Target	Weight	F1	F2	F3	F4
$case_1$	FTP guess	One to many horizontal scan	int/ext	any:any	8.50	0	0	0	0
		One to many horizontal scan	int/ext	any:any	8.50				
		Info FTP-bad login	int	ftpserver:ftpport	11.43				
		Info FTP-bad login	int	ftpserver:ftpport	11.43				
$case_2$	Webapp activity	One to many horizontal scan	int/ext	any:any	8.50	42.9	50.0	25.8	28.5
		TCP portscan	int/ext	any:any	8.57				
		SNMP trap tcp	int/ext	any:any	11.85				
		SNMP AgentX/tcp request	int/ext	any:any	11.70				
$case_3$	Dagger	TCP portscan	int/ext	any:any	8.57	71.4	83.3	43.9	48.5
		SNMP trap tcp	int/ext	any:any	11.85				
		SNMP AgentX/tcp request	int/ext	any:any	11.70				
		SNMP request tcp	int/ext	any:any	11.85				
		SCAN nmap XMAS	int/ext	any:any	10.65				
		BACKDOOR Dagger 1.4.0	int/ext	any:any	19.80				
$case_4$	NetSphere	SNMP trap tcp	int/ext	any:any	11.85	85.7	83.3	61.7	57.7
		SNMP trap tcp	int/ext	any:any	11.70				
		SNMP AgentX/tcp request	int/ext	any:any	11.70				
		SNMP request tcp	int/ext	any:any	11.85				
		SCAN nmap XMAS	int/ext	any:any	10.65				
		BACKDOOR NetSphere Access	int/ext	any:any	18.90				

Fig. 3. A fragment of the Case Memory.

correlated alerts that form the attack. This part is compared with the current attack in order to identify which kind of attack is occurring. The static component contains the type and a description of the attack, its response plan and other information (in the future its priority). After that the attack has been identified, this part can be used to prepare an appropriate incident report and response plan. In Figure 3, we reported a partial representation of attacks in the case memory. Notice that, in the case memory are represented in an "abstract" way. Let us explain this concept with an example. Figure 2; reports an attempt of opening a backdoor by a trojan horse. It consists of six events: a portscanning, four fingerprinting attempts and a trojan sent through the network. For each event, the report of the current attack includes: attack source and target IP addresses, sensor, and event description (i.e., the intrusion type). Let us suppose that in the past we had an event of the kind *SNMP trap tcp* with a given IP number (say 192.168.0.3) as target address. One of the current events is an *SNMP trap tcp* on a different IP number (say 172.16.113.84). It seems quite natural to consider these two events similar (they have the same event type), even if these two events are not identical (they have a different target). *Event abstraction* is the tool to find similarities without taking into account irrelevant details. In short, it consists of substituting some values as source and target with their type. For example, if the IP number 172.16.113.84 is the address of a web server, we can substitute the number with the keyword webserver. Formally, let $e = \langle event\_type, sensor, source, target \rangle$ be an event, then $Abs(e) = \langle event\_type, sensor\_type, source\_type, target\_type \rangle$ is the abstraction of e, $Type(e) = intrusion\_type$ is the event type of e. Let $I = \langle e_1, \ldots, e_n \rangle$ be an attack, then $Abs(I) = \langle Abs(e_1), \ldots, Abs(e_n) \rangle$ is the corresponding abstraction of I and $Type(I) = \langle Type(e_1), \ldots, Type(e_n) \rangle$ the corresponding sequence of event types of I. The attack abstraction mechanism could be based on a file or table representing the description of the actual network in the form of $\{host\_IP : host\_description\}$ pairs.

ENTROPY-BASED FILTERING

Alert correlation allows us to analyze complex attack descriptors rather than single events. The advantages of this method are the ability of recognizing stateful attacks (composed of several alerts) and the availability of an abstract description of a generic attack, independent from a specific network configuration. However, this fashion leads to drawbacks, as well. The correlation module does not perform any analysis on the attacks it produces, thus it produces several false positives in the output list. For this reason, we developed a filtering module, whose aim is to reduce the number of falses. An index of significance is needed to recognize trivial attacks. In our approach, we base this index on the well-known information theory. We define this index in two steps. First, we define a weight for each type of event. Second, we define a weight for a sequence of events.

The event weight is a function of the frequency: the lowest the frequency is, the highest the weight. An event that occurs frequently (e.g. an ICMP echo request) usually is not really dangerous and, thus, its detection provides us a little information. Formally, for each event type t, we compute the probability $p(t)$

(based on the occurring frequency) that such event type occurs and, thus, we can compute its entropy as follows:

$$w_t = -\log_2 p(t) \quad (1)$$

Concerning the weight of an attack, it is useful to notice that attacks can be roughly divided in two categories:

- significant event attacks: in which few relevant events are enough to recognize a particular attack pattern. To this category belong attacks based on malformed packets. IDS are very sensible to unexpected packets formats. In this case the attack could be composed of a single event (e.g. the LAND attack has the same value in the source and target fields of IP header).
- multitude event attacks: characterized by a large number of events, rather than their type. An attack can be performed with a big number of common and legal packets. This is the case of flooding DoS attacks. For example, a SYN flood DoS attack is done by sending a lot of TCP SYN packets to a web server. SYN packets are perfectly legal and they use to flow between client and server in the three-way-handshake phase. A DoS attack is detected if too many SYN packets come from spoofed sources. A system that analyzes single events would not be able to detect this kind of attacks.

Therefore, referring to these two categories, we defined two indices of significance for attacks evaluation: the *stateless (or low) information content* and the *stateful (or high) information content*. The former is the average of event weights in attack sequence, while the latter takes into account the multiplicity of each event type, which raises the score of stateful attacks. In formulas:

$$I(A)_{low} = \frac{\sum_{\forall e \in A} w_e}{N_A} \qquad I(A)_{high} = \frac{\sum_{\forall e \in A} w_e \cdot (1 + \alpha \cdot n_A(e))}{N_A} \quad (2)$$

where

- N_A = number of alert types in attack A
- $n_A(e)$ = number of alerts of type e in attack A
- α = multiplicity weight $(0 < \alpha < 1)$

Filter module uses two different threshold parameters: *LT (low threshold)* and *HT (high threshold)*, for stateless and stateful attacks information level evaluation, respectively. Attacks can be classified in attacks that we can ignore (*trivial attacks*) and attacks that we must take into account (*serious attacks*):

- $I(A)_{low} < LT$ *and* $I(A)_{high} < HT \Rightarrow$ TRIVIAL
- $I(A)_{low} \geq LT \Rightarrow$ SERIOUS (stateless)
- $I(A)_{high} \geq HT \Rightarrow$ SERIOUS (stateful)

The filter passes to the next module (the attack recognizer module) only the serious attacks. The two thresholds can be manually set by the security manager, or automatically set by the system. The filter has two techniques to set the thresholds, a run-time mode and a training mode. The first set the thresholds everytime an attack occurs, basing of the difference between the informative content of the new attack and the average informative content of the case memory. The second set the thresholds each epoch; an epoch consists of a predefined (by the security manager) number of attacks.

ATTACK IDENTIFICATION

The output of the entropic filter is a list of abstract attacks, each attack composed by a sequence of events. The filter eliminated most of false positives. Now, the attack identification module has to retrieve past attacks similar to the attack that passed the filter. The retrieval can be achieved by means of an appropiate similarity function. In this paper, we propose the following four:

Definition 1: Let $I_c = \langle e_{c,1}, \ldots, e_{c,n} \rangle$ be the current attack, let $I_k = \langle e_{k,1}, \ldots, e_{k,n} \rangle$ be the attack of the k-th case in the case memory, let $Type(I_c)$ and $Type(I_k)$ be the sequence of event types of I_c and I_k, respectively. Let $n_t^{(c)}$ ($n_t^{(k)}$) be the number of how many times the event type t occurs in $Type(I_c)$ ($Type(I_k)$). Let

$$m_t^{(c)} = \begin{cases} 1 \text{ if } t \in Type(I_c) \\ 0 \text{ otherwise} \end{cases} \quad m_t^{(k)} = \begin{cases} 1 \text{ if } t \in Type(I_k) \\ 0 \text{ otherwise} \end{cases}$$

be boolean functions that are true when I_c (I_k respectively) has at least an event whose type is t. Let w_t the entropy of the event type t. Then:

$$F_1(I_c, I_k) = \frac{\sum_{\forall t \in Type(I_c)} \min(n_t^{(c)}, n_t^{(k)})}{\sum_{\forall t \in Type(I_c)} n_t^{(c)}}$$

$$F_2(I_c, I_k) = \frac{\sum_{\forall t \in Type(I_c)} \min(m_t^{(c)}, m_t^{(k)})}{\sum_{\forall t \in Type(I_c)} m_t^{(c)}}$$

$$F_3(I_c, I_k) = \frac{\sum_{\forall t \in Type(I_c)} w_t \cdot \min(n_t^{(c)}, n_t^{(k)})}{\sum_{\forall t \in Type(I_c)} w_t \cdot n_t^{(c)}}$$

$$F_4(I_c, I_k) = \frac{\sum_{\forall t \in Type(I_c)} w_t \cdot \min(m_t^{(c)}, m_t^{(k)})}{\sum_{\forall t \in Type(I_c)} w_t \cdot m_t^{(c)}}$$

$F_1(I_c, I_k)$ depends on how many abstract events I_c shares with I_k. It is the ratio between the events that I_c shares with I_k (with their multiplicity) over the number of events in I_c. On the other hand, $F_2(I_c, I_k)$ depends on the number of event types shared by I_c and I_k. In other words, it considers one event per type. For example, let $Type(I_c) = \langle A, B, B, C \rangle$ the current attack and $Type(I_k) = \langle A, B, C, D \rangle$ the $k-th$ attack in the case memory, where A, B, C, D are the event types of the corresponding events. When we apply F_1, we obtain 0.75, as result; otherwise, applying F_2 we obtain 1. This is due to the fact that it counts each event type once. Notice that, F_3 (F_4) is similar to F_1 (F_2), but it also takes into account the entropy of each event type. As a consequence, these functions have a discriminating power better than F_1 and F_2.

Definition 2: Let I_c be the current attack, let KB=$\{I_1, \ldots, I_h\}$ be a set of past attacks, then $Sim_x(I_c, KB) = \overline{I} \in KB$ is the most similar past attack of I_c (according to the similarity function $F_x(.,.)$), and it is defined as follows:

$$F_x(I_c, \overline{I}) = \max_{I_i \in KB} F_x(I_c, I_i) \quad (3)$$

According to this definition, the procedure to find the most similar past attack is based on pattern matching. When a new attack occurs, the pattern abstracted by this attack (i.e., the abstract sequence of alerts) is compared with patterns in the case memory (the abstract sequence of past alerts) until a match is found. Consider the attack reported in Figure 2 and the case memory depicted in Figure 3. Let us suppose to use $F_1(.,.)$ as similarity function, then we obtain that $F_1(I_c, Case1) = 0$, $F_1(I_c, Case2) = 0.5$, $F_1(I_c, Case3) = 0.67$, and $F_1(I_c, Case4) = 0.83$. Therefore, we select the attack $Case4$ as the most similar of the current one. On the other hand, applying $F_2(.,.)$, we obtain $F_2(I_c, Case1) = 0$, $F_2(I_c, Case2) = 0.5$, $F_2(I_c, Case3) = 0.67$, and $F_2(I_c, Case4) = 0.83$ Finally, let us consider the weight in Figure 3. Applying the other retrieval functions, we obtain $F_3(I_c, Case1) = 0$, $F_3(I_c, Case2) = 0.44$, $F_3(I_c, Case3) = 0.63$, and $F_3(I_c, Case4) = 0.88$ and $F_4(I_c, Case1) = 0$, $F_4(I_c, Case2) = 0.44$, $F_4(I_c, Case3) = 0.63$, and $F_4(I_c, Case4) = 0.88$.

RESPONSE PLAN

The response plans we have used for our experiments are compliant with CERT recommendation and ISO17799. Indeed, each plan unfolds in three successive phases: in the first phase it gathers details of the current attack (incident assessment); in the second phase it improves the security of the network; in the last one it restores the normal status. As we can see in Figure 5, the first phase of the response plan extracts information related to the current attack: the name of the process, the numbers of the ports, the file that opens not reliable connections, etc. The second phase of the plan improves the security of the system, killing bad processes, updating system and security tools, installing patches, closing bad connections or suspicius ports, etc. Finally, the third phase aims to reach a normal condition using backup copies to restore services, software and files.

From the point of view of our system, the retrieved response plan simply consists of a sequence of abstract actions. Each abstract action must be translated in a concrete action. In order to do that, for each abstract action we have a set of concrete actions each of them related to a given host, operating system, environment, etc. Therefore, for each concrete action, we have appropiate preoconditions to be satisfied. In other words, a concrete action can be used as translation of a given abstract if and only if its preconditions are satisfied. As a consequence, a concrete plan consists fo a sequence of concrete actions with true preconditions. Let us explain this by an example. In Figure 4, the first column has the abstract actions of the plan, the second column has the related concrete action and the third column has the preconditions. Indeed, the first two actions aim to check the current process list comparing it with a reliable list, in order to detect bad processes. In the second column, we have the concrete actions for Windows OS and Linux OS. The system selects the second one, because the machine runs a Linux OS. The same process is applied to the other actions and the result plan is in Figure 5. As we said our plans respect these three phases suggested by CERT, but the adaptation algorithm does not consider them.

When a new attack occurs, the retrieval module searches for the closest past case in the case memory, and returns its linked plan. This plan is related to the similar attack, hence it is useful for the security manager: it consists of several advices to block the attack and restore the normal status. The plan is executed semi-automatically: before executing each action, the system asks a confirmation to the security manager, who can accept or deny the execution. If he denies the execution, he can change the action correct and execute it. After the execution, the new case with the correct plan are retained in the case memory: in this way, the system learns from new cases to improve the effectiveness.

CONCLUSIONS

Our work deals with an incident response system and the securiy manager activity. In this paper, we presented a tool for detecting and identifying complex attacks (a sequence of alerts that have the same goal), presenting a semi-automatic response plan to the security manager.

It is an innovative system because it covers all the phases of an attack response security system: from identification to response. Attack identification has been recognized as one of the most crucial activities, if we want to have a possibility of responding appropriately to an attack (e.g., see [11], [4]). Obviously, the attack identification process strongly depends on the previous experience of the security manager, but our system may help him with its identification system, proposing appropriate response plans and automating operations that can be.

Furthermore, every time a new attack is detected, the security manager must learn it. This model of the identification process relies on the observation that, as can be noticed in the DARPA experiments [12], it is infrequent to have twice the same attack, but it is very common to have several "similar" attacks. These three characteristics, experience-based reasoning, similarity-based retrieval and learning are the base of the proposed tool. Furthermore, our system is able to propose response plans based on responsive actions adopted for past similar attacks. Even if the kind of attack is new, the system is still able to propose, to the security manager, a response plan. If this plan is evaluated as appropiate, it can also be retained to improve system experience. Where it is possible, plans are automat-

Abstract Actions	Concrete Actions	Preconditions
check the process list	taskmgr.exe ps -aux >> proclist.txt	OS=Microsoft Windows OS=Linux
compare the process list to the reliable one	diff proclist.txt reliabprocs.txt >> badprocs.txt winmerge.exe proclist.txt reliabprocs.txt >> badprocs.txt	OS=Linux OS= Microsoft Windows
check the connection list	netstat >> ports.txt	OS=Microsoft Windows, Linux
compare the connection list to the reliable one	diff network.txt reliabnet.txt >> badconn.txt fc network.txt reliabnet.txt >> badconn.txt	OS=Microsoft Windows, Linux OS=Microsoft Windows
check the register list and compare it to \ the reliable list	diff regedit.txt reliabregedit>> modifiedKeys.txt	OS=Microsoft Windows
kill illegal processes	badprocs.txt >> kill -9 End Process	OS=Linux OS=Microsoft Windows
remove vulnerabilities installing patches, \ updating systems	update windows	internet connection or update-CD/DVD
remove modified keys listed in \ modifiedkeys.txt	yum update update explorer update firefox update antivirus	Fedora Linux with Internet connection MS IExplorer Mozilla Firefox antivirus present
configure the firewall to reject traffic from \ the exploited ports	ports.txt >>access-list 101 deny tcp eq 25 ports.txt >> iptables -p –dport -j REJECT,	CISCO firewall iptables, Linux
restore the normal status of the network	use backup copies to restore services use backup copies to restore corrupted softwares use backup copies to restore damaged files	backup and backup utility backup and backup utility backup and backup utility

Fig. 4. The response plan linked to the case4.

Response plan
ps -aux >> proclist.txt diff proclist.txt reliabprocs.txt >> badprocs.txt lsof >> ports.txt diff ports.txt reliabports.txt >> badports.txt
badprocs.txt >> kill -9 yum update update firefox update antivirus badports.txt >> iptables -p –dport -j REJECT
use backup copies to restore services use backup copies to restore corrupted softwares use backup copies to restore damaged files

Fig. 5. The adapted plan.

ically executed, after the security manager validation. From preliminary experiments done on the whole tool, it arises that a tool based on the experience, and capable of learning, capturing what a security manager usually does is quite hard to realize, but the preliminary results are promising. As future work, we will present experimental results.

References

[1] F. Valeur and G. Vigna and C. Kruegel and R. Kemmerer, *A Comprehensive Approach to Intrusion Detection Alert Correlation*, IEEE Transaction on Dependable and Secure Computer,2004,Volume 1,number 3, pages 146–169,July–September.

[2] C. Kruegel and G. Vigna, *Anomaly Detection of Web-based Attacks*, Proceedings of the 10^{th} ACM Conference on Computer and communications security, 2003,pages 251–261,Washington D.C.,October,ACM Press

[3] M. Esmaili and R. Safavi-Naini and B. Balachandran and J. Pieprzyk, *Case-Based Reasoning for Intrusion Detection*, Proceedings of the 12^{th} Annual Computer Security Applications Conference, 1996,pages 214–223,December

[4] CERT Coordination Center, *Responding to Intrusions*, 2001,http://www.cert.org/security-improvement/modules/m06.html

[5] M. Roesch, *Snort*, 1998, http://www.snort.org/

[6] Tripwire Inc., *Tripwire*, http://www.tripwire.com/

[7] Cisco Systems, Inc., *Cisco Security Monitoring, Analysis and Response System*, http://www.cisco.com/en/US/products/

[8] Arcsight, Inc., *Arcsight Security Manager*, http://www.arcsight.com

[9] E. Yilmaz and S. Stoecklin and D. G. Schwartz, *Toward a Generic Case-Based Reasoning Framework Using Adaptive Software Architectures*, IKE,2003,pages 512–514,Las Vegas,Nevada,June

[10] N. Amani and M. Fathi and M. Dehghan, *A Case-Based Reasoning Method for Alarm Filtering and Correlation in Telecommunication Networks*, Proceedings of the Electrical and Computer Engineering, 2005,pages 2182–2186,Canada,May

[11] International Organization for Standardization, *ISO/IEC 17799 – BS7799 — Information technology: Code of practice for information security management*, 2002,http://www.iso17799software.com/

[12] MIT Lincoln Laboratory, DARPA, *DARPA Intrusion Detection Evaluation Data Sets*, 1999,http://www.ll.mit.edu/IST/ideval/index.html

[13] M. Nick and B. Snoek and T. Willrich, *Supporting the IT Security of eServices with CBR-Based Experience Management*, Proceedings of 5^{th} International Conference on Case-Based Reasoning Research and Development (ICCBR 2003), 2003,volume 2698,Trondheim,Norway,June

[14] P. Porras and P. Neumann, *EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances*, Proceedings of the 1997 National Information Systems Security Conference, 1997

[15] N. Ye and B. Harish and T. Farley, *Attack profiles to derive data observations, features, and characteristics of cyber attacks*, Information Knowledge Systems Management, 2005/2006,volume 5,IOS Press

[16] J. P. Wack, *Establishing a Computer Security Incident Response Capability (CSIRC)*, Computer Systems Laboratory, National Institute of Standards and Technology, 1991,NIST Special Publication 800-3,November

[17] R. Barruffi and M. Milano and R. Montanari, *Planning for security management*, Intelligent Systems IEEE,Volume 16,Jan-Feb,2001,pages 74–80

[18] A. Aamodt and E. Plaza, *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System approaches*, AI Communications,1994,volume 7,number 1,pages 39–59

Special Session on Parallel and Grid Computing for Optimization

A GRID-BASED HYBRID CELLULAR GENETIC ALGORITHM FOR VERY LARGE SCALE INSTANCES OF THE CVRP

Bernabé Dorronsoro
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: bernabe@lcc.uma.es

Daniel Arias
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: bernabe@lcc.uma.es

Francisco Luna
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: flv@lcc.uma.es

Antonio J. Nebro
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: antonio@lcc.uma.es

Enrique Alba
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: eat@lcc.uma.es

KEYWORDS
Cellular genetic algorithm, grid computing, VRP

ABSTRACT
This work presents a hybrid genetic algorithm (GA) for solving the largest existing benchmark instances of the capacitated vehicle routing problem (CVRP). The population of the algorithm is structured by following two classical parallelization models for GAs: coarse- and fine-grained. Indeed, the proposed model is a distributed GA (coarse-grained) in which each island is a cellular GA (fine-grained). It has been called PEGA (Parallel cEllular Genetic Algorithm). PEGA has been built on top of ProActive and it has been executed on a grid platform composed of more than 100 machines so as to reduce the computation time. The results show that, for many of the considered instances, PEGA improves the best results reported by any existing algorithm in the literature.

INTRODUCTION

The vehicle routing problem (VRP) (Dantzing and Ramster 1959) lies in minimizing the cost of a fleet of vehicles serving a set of customers from a unique depot (Fig. 1). Reducing this cost means minimizing the number of used vehicles and the length of their routes as well. This problem is of great interest due to two main facts. On one hand, the VRP is very interesting from an academic viewpoint because of its complexity (it is an NP-hard problem (Lenstra and Kan 1981)), the constraints it includes, and the many different existing versions. On the other hand, it has a direct application to the real world since it can be used by both large logistic enterprises and small local delivering companies. Indeed, using computer methods in such a business could often lead important cost savings to be reached. In many cases, these savings mean 20% of the total cost of the product (Toth and Vigo 2001).

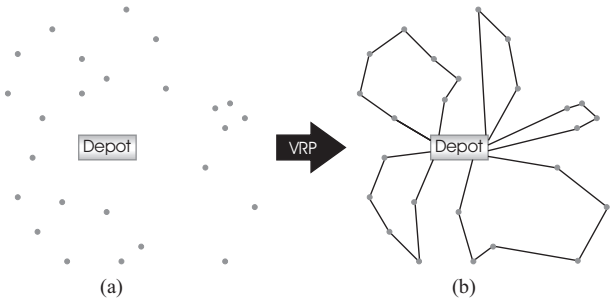


Fig. 1. The Vehicle Routing Problem consists in serving a set of geographically distributed customers (points) from a depot (a) using the minimum cost routes (b)

In the recent history of VRP, a constant evolution exists in the quality of the methodology used for solving the problem. These techniques comprise exact algorithms as well as heuristic methods. However, due to the complexity of the problem, there is no exact method capable of solving instances with more than 50 customers (Toth and Vigo 2001; Golden et al. 1998). It is also clear that generic heuristics cannot compete, in terms of solution quality, against current techniques such as those described in (Toth and Vigo 2001; Cordeau et al. 2005), which are specifically developed for solving VRP. Moreover, the potential search of some of these modern techniques, e.g. genetic algorithms (GAs), is not still fully exploited, especially when combining them with effective local search mechanisms. All these considerations could allow us to improve the search capability of a given algorithm.

This work is therefore focussed on GAs. Due to their population-based approach, GAs are very suitable for parallelization because their main operations (i.e., crossover, mutation, local search, and function evaluation) can be performed independently on different individuals. As a consequence, the performance of population-based algorithms is specially improved when run in parallel. Two parallelizing strategies are especially relevant for population-based algorithms: (1) parallelization of computation, in which the operations commonly

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

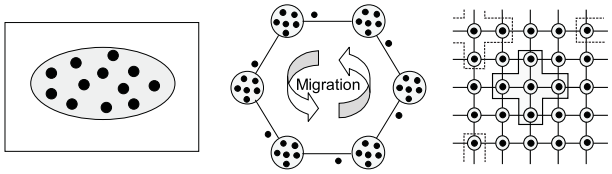


Fig. 2. Panmictic (a), distributed (b), and cellular (c) GAs

applied to each individual are performed in parallel, and (2) parallelization of population, in which the population is split in different parts, each one evolving in semi-isolation (individuals can be exchanged between subpopulations). Among the most widely known types of structured GAs, the *distributed* (dGA) (or coarse-grain) and *cellular* (cGA) (or fine-grain) algorithms are very popular optimization procedures (see Fig. 2). The parallelization of the population strategy is specially interesting since it does not only allow to speed up the computation, but also to improve the search capabilities of GAs (Alba and Tomassini 2002; Cantú-Paz 2000).

We propose in this work a new parallel cGA called PEGA (Parallel cEllular Genetic Algorithm). PEGA adopts all the parallelization strategies described above: the population is divided into several islands (dGA), the population of each island being structured by following the cellular model (cGA). Furthermore, each cGA implements a master/slave approach (parallelization of the computation) in order to compute the most costly operations applied to their individuals. Periodically, cGAs exchange information (migration) with the goal of inserting some diversity into their populations, thus avoiding them falling into local optima. By using such structure in the population, we keep a good balance between exploration and exploitation, thus improving the capability of the algorithm to solve complex problems (Alba and Tomassini 2002; Alba and Dorronsoro 2007). Additionally, cGAs in PEGA are hybridized with a local search method which is applied to every newly generated solution.

When solving very large problem instances, even heuristic methods like PEGA require a large amount of computational resources. This is exactly the context in which PEGA was developed because it is ultimately aimed at solving the largest instances of CVRP (a subclass of VRP): the VLSVRP benchmark (Li et al. 2005). With this goal in mind, PEGA has been enabled to run in *grid* computing platforms. This way, it has been implemented in ProActive (ProActive 2007), a Java GRID middleware library for an easy programming in *grid* environments. ProActive provides mechanisms for creating and managing collaborating objects (called *active objects*) which can be executed on remote machines. The remote access to an active object is carried out transparently by ProActive by using Java RMI.

The main contribution of this work is therefore the design of a new hybrid cGA which runs in parallel on *grid* computing platforms. This algorithm has been used to solve the largest known benchmark instances of the CVRP (Li et al. 2005) and it is able to improve the best known solutions computed by an optimization algorithm for most of the studied instances.

The paper is structured as follows. In the next section, we mathematically define the CVRP problem. Next, we describe PEGA, our proposal for solving large instances of the CVRP. Finally, the parameterization of the algorithm, the benchmark used, and the results obtained, as well as our conclusions and the main lines of future work are given in the two last sections.

THE VEHICLE ROUTING PROBLEM

The VRP can be defined as an integer programming problem which falls into the category of NP-hard problems (Lenstra and Kan 1981). Among the different variants of VRP we work here with the Capacitated VRP (CVRP), in which every vehicle has a uniform capacity of a single commodity. The CVRP is defined on an undirected graph $G = (\vec{V}, \vec{E})$ where $\vec{V} = \{v_0, v_1, \ldots, v_n\}$ is a vertex set and $\vec{E} = \{(v_i, v_j)/v_i, v_j \in \vec{V}, i < j\}$ is an edge set. Vertex v_0 stands for the *depot*, and it is from where m identical vehicles of capacity Q must serve all the *cities* or *customers*, represented by the set of n vertices $\{v_1, \ldots, v_n\}$. We define on E a non-negative *cost*, *distance* or *travel time* matrix $C = (c_{ij})$ between customers v_i and v_j. Each customer v_i has non-negative demand of goods q_i and drop time δ_i (time needed to unload all goods). Let be $\vec{V_1}, \ldots, \vec{V_m}$ a partition of $\vec{V}$, a route $\vec{R_i}$ is a permutation of the customers in $\vec{V_i}$ specifying the order of visiting them, starting and finishing at the depot v_0. The cost of a given route $\vec{R_i}$ =$\{v_0, v_1, \ldots, v_{k+1}\}$, where $v_j \in \vec{V}$ and $v_0 = v_{k+1} = 0$ (0 denotes the depot), is given by:

$$\mathrm{Cost}(\vec{\mathrm{R}_\mathrm{i}}) = \sum_{\mathrm{j}=0}^{\mathrm{k}} \mathrm{c_{j,j+1}} + \sum_{\mathrm{j}=0}^{\mathrm{k}} \delta_\mathrm{j} \ , \qquad (1)$$

and the cost of the problem solution ($\vec{S}$) is:

$$F_{\mathrm{CVRP}}(\vec{S}) = \sum_{i=1}^{m} \mathrm{Cost}(\vec{\mathrm{R}_\mathrm{i}}) \ . \qquad (2)$$

The CVRP lies in determining a set of m routes (i) of minimum total cost (see Equation 2); (ii) starting and ending at the depot v_0; and such that (iii) each customer is visited exactly once by exactly one vehicle; subject to the restrictions that (iv) the total demand of any route does not exceed Q ($\sum_{v_j \in \vec{R_i}} q_j \leq Q$); and (v) the total duration of any route is not larger than a preset bound D ($\mathrm{Cost}(\vec{\mathrm{R}_\mathrm{i}}) \leq \mathrm{D}$).

PEGA: PARALLEL CELLULAR GENETIC ALGORITHM

PEGA is a parallel GA in which the population is structured at two levels and in two different ways. As it can be seen in Fig. 3, the population is decentralized in a first

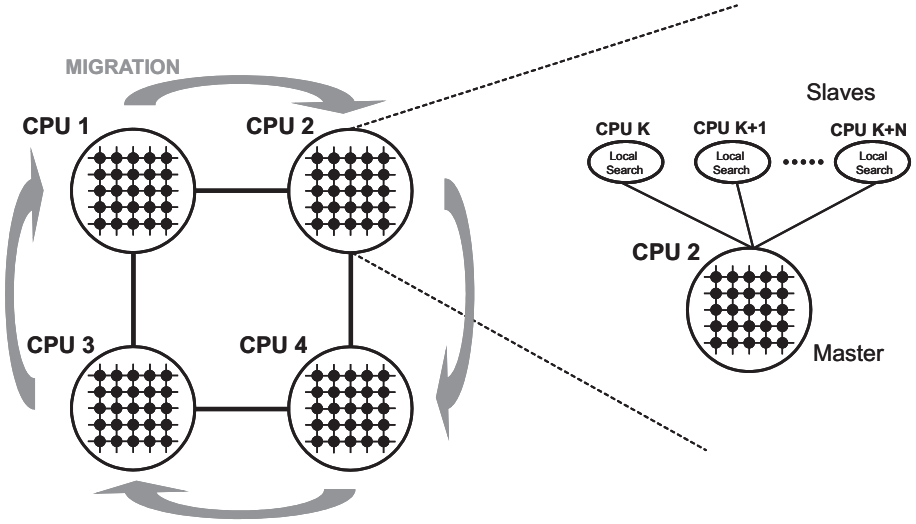


Fig. 3. Structure of PEGA

level by using the island model. The topology of these islands is an unidirectional ring so that migrations only occur between immediate neighbors. At the second level, each island is a cGA (see Section) in which the population follows a fine-grained parallelization strategy. Furthermore, each cGA is a master process that sends individuals to slave processes where they undergo the local search method, the task demanding the higher computational costs in the reproductive cycle of the cGA. This model and its parallel implementation in clusters were initially proposed in (Alba 1999).

Cellular Genetic Algorithm

This section describes the cGA used in each island of PEGA. Algorithm 1 shows its pseudocode. The population is structured in a 2D toroidal grid where a neighborhood structure is defined. The algorithm operates iteratively on each individual of the population (lines 3 and 4). The individuals can only interact with their nearby neighbors (line 5) and the parents are therefore chosen from the neighborhood of the current individual (line 6) with a given criterion. The recombination and mutation operators are applied in lines 7 and 8 with probabilities P_c and P_m, respectively. After that, the resulting individual undergoes a local search phase (line 9), and next its fitness value is computed (i.e., the cost of the represented solution). The local search method is explained below. Finally, the best individual between the current and the offspring is placed in the equivalent position of an auxiliary population (line 11).

Algorithm 1 Pseudocode of a cGA

```
1:  proc Steps_Up(cga)        //Algorithm parameters in 'cga'
2:  while not Termination_Condition() do
3:  for x ← 1 to WIDTH do
4:  for y ← 1 to HEIGHT do
5:   n_list←Get_Neighborhood(cga,position(x,y));
6:   parents←Selection(n_list);
7:   aux_indiv←Recombination(cga.Pc,parents);
8:   aux_indiv←Mutation(cga.Pm,aux_indiv);
9:   aux_indiv←Local_Search(cga.Pl,aux_indiv);
10: Evaluate_Fitness(aux_indiv);
11: Replacement(position(x,y),aux_indiv,cga,aux_pop);
12: end for
13: end for
14: cga.pop←aux_pop;
15: Update_Statistics(cga);
16: end while
17: end_proc Steps_Up;
```

Once the reproductive cycle is applied to all the individuals of the population, the current population is replaced by the auxiliary one (line 14) and we then calculate some statistics (line 15). This loop is repeated until a termination condition is met.

Next, we detail some major issues concerning the implementation of the cGA used:

- **Individual representation.** We have used GVR (*Genetic Vehicle Representation*) (Pereira et al. 2002) for encoding the individuals. GVR uses a permutation of integer numbers which contains both customers and route splitters (delimiting different routes). Each route is composed of the customers between two route splitters in the individual. No unfeasible individuals are allowed so when either the maximum capacity of the vehicles or the maximum route length are exceeded, a repair procedure is executed so that it splits routes into two or more different subroutes which do not violate any constraint. This repair procedure is very fast and it also introduces a high level of diversity into the population. This represents an important point since in our previous studies (not using GVR) the population diversity was lost at the end of the evolution (Alba and Dorronsoro 2004; Alba and Dorronsoro 2007).
- **Generation of the initial population.** The individuals of the initial population are randomly generated, and then they are modified as shown in (Pereira et al. 2002) with the goal of obtaining feasible solutions.
- **Recombination operator.** PEGA uses the *generic crossover* originally proposed in (Pereira et al. 2002). The main feature of this operator is to promote diversity in the population. This is a very important feature since in our previous experiences addressing this problem (Alba and Dorronsoro 2007; Alba and Dorronsoro 2004; Alba

Algorithm 2 Pseudocode of the recombination operator

// Let I_1 and I_2 be the chosen parents from the neighborhood;
Choose a random subroute $SR = \{a_1, \ldots, a_n\}$ de I_2
Find the customer $c \notin SR$ geographically closest to a_1
Remove all the customers from I_1 that are included in SR
The offspring is obtained after inserting SR into the genetic material of I_1 so that a_1 is placed just after c

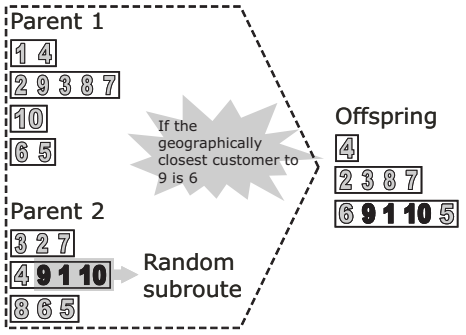


Fig. 4. Recombination operator used: *Generic crossover*

and Dorronsoro 2006), the entire population converges towards the same local optimum many times. This operator is somewhat unusual because the newly generated offspring does not only include genetic material from the two parents but also random components, as shown in Fig. 4. Algorithm 2 outlines the crossover operator used.

• **Mutation.** The mutation phase is composed of four different mutation operators which are applied with different probabilities (only one of them is applied each time), as proposed in (Pereira et al. 2002). Using these four mutation procedures allows us to modify the itinerary of a route, to move customers between routes, and to add or remove routes. They are (see Fig. 5):

– **Swap.** It swaps the position of two randomly chosen customers (belonging to the same route or not).

– **Inversion.** It reverses the visiting order of the customers between two randomly selected points of the permutation. In this case, all the customers must be in the same route.

– **Insertion.** It selects a gene (either customer or route splitter) and inserts it in another randomly selected place of the same individual.

– **Dispersion.** It is similar to the Insertion operator, but it is applied to a subroute (set of customers) rather than to a single customer.

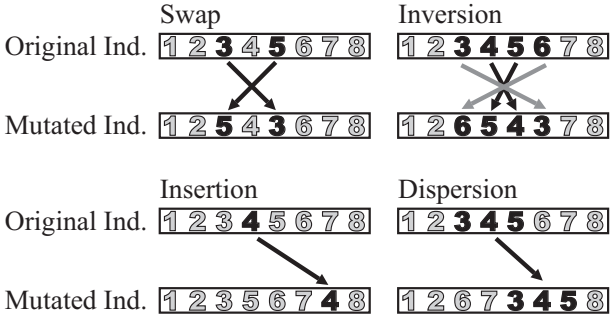


Fig. 5. The mutation operators

• **Local search.** The local search method applies up to 50 steps of *1-Interchange* (Osman 1993) and then up to 50 steps of *2-Opt* (Croes 1958) to each route of the solution reached by *1-Interchange*. These values were set after a tuning process. The *1-Interchange* method lies in interchanging a customer from a route by other customer belonging to other route, or inserting a customer from a route in a different route. On the other hand, *2-Opt* always works on one single route. It removes two edges of one route and connects the customers in the possible way (see Fig. 6). Since these two local search methods are deterministic, the search stops if no improvements have been reached in one single step. This will allow us to largely reduce the execution times.

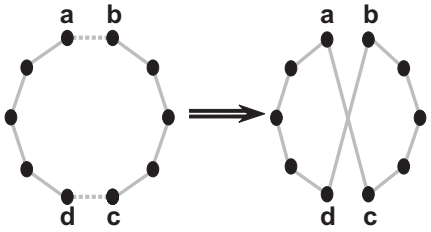


Fig. 6. The 2-Opt operator

EXPERIMENTATION

Recently, Li, Golden and Wasil presented in (Li et al. 2005) a new set of instances for the CVRP which are mainly characterized by the high number of customers used. This set of problems was called VLSVRP or *Very Large Scale VRP*. The size of the proposed instances in VLSVRP ranges between 560 and 1200 customers, whereas the most widely used and accepted benchmarks in the research community up to now were composed of instances between 50 and 199 customers in the CMT case (Christofides et al. 1979), or problems between 200 and 483 customers in (Golden et al. 1998). Some additional important characteristics of VLSVRP are: it has been generated by using an instance generator, thus easing the creation of instances of larger size (the reader is referred to (Li et al. 2005) for more details on the instance generator); the generated instances are geometrically symmetric with a circular pattern, which allows the best solution to be visually estimated; and the instances of the VLSVRP all include constraints on the maximum length of the routes.

PEGA has been designed with the aim of being used to solve the VLSVRP instances. Because of the complexity and large size of these problems, PEGA has been executed on a grid composed of up to 125 heterogeneous computers (PCs and Sun workstations/servers). The algorithm is implemented in Java, using ProActive to manage all the grid related issues. The parameterization of PEGA used for the experiments is detailed in Table I. Concretely, PEGA is composed of 4 islands arranged in a unidirectional ring. Each island operates the cGA described above in Section , using a square toroidal grid of

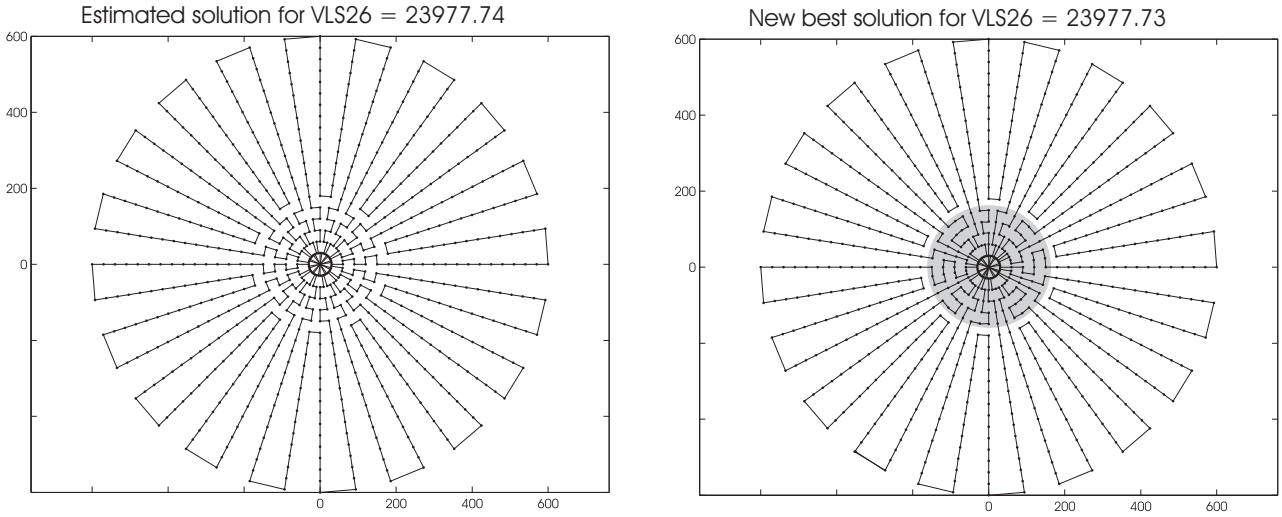


Fig. 7. New best solution for VLS26. Differences between the new and the previous solution can be noticed in the shadowed area

TABLE I: Parameterization used in PEGA

Population size	Islands: 100 Individuals (10×10) Total: 400 Individuals (4 islands)
Neighborhood	NEWS
Parent Selection	Binary tournament + Current Individual
Recombination	*generic crossover* (Pereira et al. 2002), $p_c = 1.0$
Mutation	Swap ($p_{\text{int}} = 0.05$), Inversion ($p_{\text{inv}} = 0.1$), Insertion ($p_{\text{ins}} = 0.05$), and Dispersion ($p_{\text{disp}} = 0.15$)
Replacement	Replace if better
Local Search	1-Interchange + 2-Opt, 50 optimization steps each
Migration Frequency	Every 10^4 evaluations
Stopping Condition	500,000 evaluations in each island

10×10 individuals. In the reproductive cycle, one parent is chosen by binary tournament in the neighborhood of the current individual (this neighborhood is composed of the individuals in the North, East, West, and South – NEWS). The other parent is the current individual itself. The two parents are always recombined ($p_c = 1.0$) by using the *generic crossover* operator. The resulting individual undergoes mutation by one of the operators Swap, Inversion, Insertion, and Dispersion with probabilities 0.05, 0.1, 0.05, and 0.15, respectively (Pereira et al. 2002). Next, the local search method is applied to the mutated individual. As explained before, the method firstly executes 50 steps of *1-Interchange* (exploring combinations of customers among different routes) and then it applies 50 steps of *2-Opt* so as to optimize separately each newly generated route. The resulting offspring replaces the current individual in the population if the former has a better fitness value than the latter.

In the proposed implementation, migration takes place every 10^4 evaluations. At each migration operation, the islands send their best individual to their immediate neighbor. When a subpopulation receives a migrant, it replaces the worst individual in the local population.

Table II presents the results of PEGA for the VLSVRP benchmark (the best result for each instance is marked in **boldface**). The table includes the name of the studied instances, their sizes (number of customers to be served), the best known solution for each instance (BNS) (Li et al. 2005), the best and average solutions found by PEGA, and the average execution time it takes. The fitness values of the solutions represent the cost in terms of the global distance traversed by the vehicles.

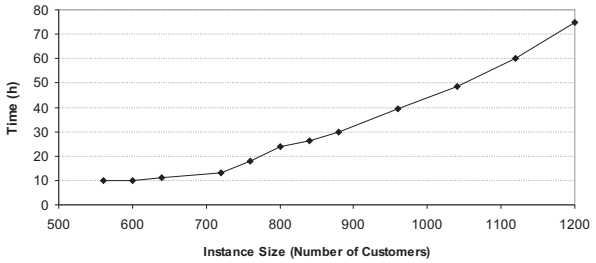


Fig. 8. Computation times of PEGA for different problem sizes

The values in Table II have been obtained after performing four independent runs of the smaller instances (VLS21 to VLS25) and two independent runs of the remaining instances. This small number of independent runs is due to the very long computational times required by PEGA to solve the problem, which ranges from 10 hours for the smaller instances to 72 hours in the case of VLS32. This is motivated by two main issues: (i) on one hand, the local search is a very high computational demanding task, which grows almost exponentially with the size of the solved instance; (ii) on the other hand, since the optimal solution is unknown, the stopping condition is to reach a preprogrammed number of function evaluations, and this number has to be large enough so that PEGA shall be able to find or even overcome the best known solution. Figure 8 shows an evolution of the computational times with the size of the instances. As it can be seen, the grown curve of these times over the increasing problem size is more than linear.

It is remarkable the accurate results despite the low

TABLE II: Results of PEGA for VLSVRP

Instance	Size	BNS	Best	Average	Time (h)
VLS21	560	**16212.83**§	**16212.83**	$16212.83_{\pm 4.42e-4}$	10.08
VLS22	600	**14641.64**†	14652.28	$14755.90_{\pm 98.88}$	10.08
VLS23	640	**18801.13**§	**18801.13**	$18801.13_{\pm 4.32e-6}$	11.28
VLS24	720	**21389.43**§	**21389.43**	$21389.43_{\pm 7.63e-6}$	13.20
VLS25	760	**17053.26**§	17340.41	$17423.42_{\pm 72.10}$	18.00
VLS26	800	23977.74§	**23977.73**	$23977.73_{\pm 3.49e-5}$	23.76
VLS27	840	**17651.60**†	18326.92	$18364.57_{\pm 37.66}$	26.40
VLS28	880	**26566.04**§	**26566.04**	$26566.04_{\pm 1.33e-6}$	30.00
VLS29	960	**29154.34**§	**29154.34**	$29154.34_{\pm 4.24e-5}$	39.60
VLS30	1040	**31742.64**§	31743.84	$31747.51_{\pm 3.67}$	48.72
VLS31	1120	**34330.94**§	**34330.94**	$34331.54_{\pm 0.60}$	60.00
VLS32	1200	**36919.24**§	37423.94	$37431.73_{\pm 7.79}$	74.88

§ Visually estimated solution (Li et al. 2005); † ORTH (Li et al. 2005)

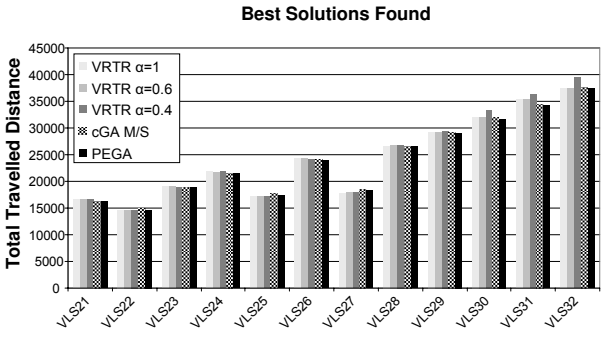


Fig. 9. PEGA vs. the state-of-the-art for solving the VLSVRP instances

TABLE III: Difference (%) between the best known solution and the results of PEGA and the algorithms of the state-of-the-art

Instance	VRTR (α value) (1.0)	(0.6)	(0.4)	cGA M/S	PEGA
VLS21	2.41	2.56	3.25	0.02	**0.00**
VLS22	**0.07**	0.10	0.19	2.17	**0.07**
VLS23	1.09	1.56	0.20	**0.00**	**0.00**
VLS24	1.85	1.06	2.54	$5.61e-3$	**0.00**
VLS25	0.58	0.65	**0.55**	3.53	1.68
VLS26	0.88	0.93	0.13	0.05	**0.00**
VLS27	**0.97**	1.61	1.42	4.83	3.83
VLS28	0.15	0.82	0.83	**0.00**	**0.00**
VLS29	0.09	0.10	0.85	0.02	**0.00**
VLS30	0.74	0.69	4.74	0.64	**3.78e − 3**
VLS31	3.02	2.98	5.85	0.35	**0.00**
VLS32	1.36	**1.33**	6.76	2.02	1.37
Average	1.10	1.20	2.28	1.14	**0.58**

number of independent runs which have been carried out. Indeed, PEGA reaches the best known solution in 7 out of the 12 studied instances. Furthermore, PEGA has been able to improve the best known solution for VLS26. Even though the difference is very small, they represent very different solutions, as it can be seen in the shadowed area of Fig. 7. We also want to note that the best known solutions for all the instances have been visually estimated by using the geometric properties of the problems, and no algorithm was able to find them up to this work. In the case of the instances VLS22 and VLS27, the solutions were found by different RTR (*Record-To-Record*) algorithms that were tried in the development phase of VRTR (an enhanced version of RTR) (Li et al. 2005). However, neither details on these algorithms nor references to related works are given in (Li et al. 2005) for further reading on this topic. In Table II, ORTR means Other RTR algorithms.

Figure 9 shows a comparison between the best solutions obtained by PEGA, a master/slave cGA (called cGA M/S) with the same configuration used in the cGAs of the islands of PEGA (except for the termination condition, which is set to 200,000 evaluations in this case), and the three algorithms which are the state-of-the-art for VLSVRP. These three algorithms are different parameterizations of VRTR proposed in (Li et al. 2005). As it can be seen, PEGA gets equal or lower fitness values (better results) than cGA M/S for all the tested instances. Regarding the three VRTR versions, PEGA outperforms them in all the instances but VLS25, VLS27, and VLS32. In Table III, we present the comparison among the three VRTR algorithms taken from the literature (Li et al. 2005), PEGA, and cGA M/S. The comparison is made for all the problem instances in terms of the difference (percentage) between the solution they reported and the best known solution (best values are in **boldface**). This value can be understood as a quality measure of the results obtained by the algorithms. From the results, it is noticeable that PEGA is a more robust algorithm with respect to the other compared approaches in the studied benchmark since it gets the best results in 9 out of the 12 VLSVRP instances. Additionally, PEGA obtains the best known solution in 7 instances, solutions which have never been reached by any algorithm, as stated before. If we compare PEGA with the cGA M/S we can notice that structuring the population in two different levels (fine- and coarse-grained) is highly beneficial versus using just one level of decentralization (the fine-grained model of cGA M/S) for the tested problems. Indeed, PEGA outperforms our cGA M/S for all the tested instances.

The last row in Table III shows the average of the differences between the solutions found by each algorithm and the best known solutions of the 12 VLSVRP instances. As it can be seen, PEGA gets the lowest (best) value among the four compared algorithms. Indeed, this value is half the value of the best VRTR approach ($\alpha = 1.0$).

CONCLUSIONS AND FUTURE WORKS

This work presents a very powerful algorithm for solving extremely hard instances of CVRP. The proposed algorithm, called PEGA, is a distributed GA with four islands, in which each island is in turn a cellular GA. The population is therefore structured at two levels. Additionally, and because of the complexity of the studied instances, PEGA has been parallelized using ProActive so that it can be executed on grid computing platforms. The parallelization strategy used in each island follows a master/slave scheme as well.

PEGA has been compared against the algorithms of the state-of-the-art for the benchmark of the CVRP which comprises the largest instances, the set of problems VLSVRP. As a result, PEGA not only was able to find the best results in 9 out of the 12 instances, but also it computed the best known solution for 7 instances (never found before by any algorithm, but they have been visually estimated by using their geometric features). PEGA also found a new best solution for the VLS26 instance.

As very near future work, we are working on increasing the number of independent runs of our experiments, because we want to provide the results with statistical confidence. We are also planing to reduce the computational times by increasing the size of our grid computing platform as well as by improving the algorithm to make it more efficient. In this second issue, our idea is to focus on developing new improved local search methods, because it is the most computationally costly step of the reproductive cycle.

ACKNOWLEDGEMENTS

The authors are partially supported by the Spanish Ministry of Education and Science, and by European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project, http://oplink.lcc.uma.es).

REFERENCES

Alba, E. 1999. “Análisis y diseño de algoritmos genéticos paralelos distribuidos,” Ph.D. dissertation, Universidad de Málaga, Málaga.

Alba, E. and Dorronsoro, B. 2004. “Solving the vehicle routing problem by using cellular genetic algorithms,” in *Evolutionary Computation in Combinatorial Optimization – EvoCOP04*, ser. LNCS, J. Gottlieb and G. R. Raidl, Eds., vol. 3004. Coimbra, Portugal: Springer Verlag, 11–20.

Alba, E. and Dorronsoro, B. 2006. “Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm,” *Information Processing Letters*, 98(6):225–230.

Alba, E. and Dorronsoro, B. 2007. *Engineering Evolutionary Intelligent Systems*, ser. Studies in Computational Intelligence. Springer-Verlag, ch. 13, “A Hybrid Cellular Genetic Algorithm for the Capacitated Vehicle Routing Problem”.

Alba, E. and Tomassini, M. 2002. “Parallelism and evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, 6(5):443–462.

Cantú-Paz, E. 2000. *Efficient and Accurate Parallel Genetic Algorithms*, 2nd ed., ser. Book Series on Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, vol. 1.

Cordeau, J.; Gendreau, M.; Hertz, A.; Laporte, G. and Sormany J. 2005 *New Heuristics for the Vehicle Routing Problem*. Logistics Systems: Design and Optimization, A. Langevin and D. Riopel, eds., Springer, New York, 279–297.

Christofides, N.; Mingozzi, A. and Toth, P. 1979. *Combinatorial Optimization*. John Wiley, 1979, ch. “The Vehicle Routing Problem,” 315–338.

Croes, G. 1958. “A method for solving traveling salesman problems,” *Operations Research*, 6:791–812.

Dantzing, G. and Ramster, R. 1959. The truck dispatching problem. *Management Science* 6, 80–91.

Golden, B.; Wasil, E.; Kelly, J. and Chao I.-M. 1998. *Fleet Management and Logistics*. Boston: Kluwer, 1998, ch. “The Impact of Metaheuristics on Solving the Vehicle Routing Problem: algorithms, problem sets, and computational results,” 33–56.

Lenstra, J. and Kan, A.R. 1981. “Complexity of vehicle routing and scheduling problems,” *Networks*, vol. 11, 221–227.

Li, F.; Golden, B. and Wasil, E. 2005. “Very large-scale vehicle routing: New test problems, algorithms, and results,” *Computers & Operations Research*, vol. 32, 1165–1179.

Osman, I. 1993. “Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems,” *Annals of Operations Research*, 41:421–451.

Pereira, F.; Tavares, J.; Machado, P. and Costa, E. 2002. “GVR: a new representation for the vehicle routing problem,” in *13th Irish Conference Proceedings on Artificial Intelligence and Cognitive Science (AICS)*, M. O. et al., Ed. Ireland: Springer-Verlag, 95–102.

“ProActive official web site,” http://www-sop.inria.fr/oasis/proactive/

Toth, P. and Vigo, D. 2001. *The Vehicle Routing Problem*, ser. Monographs on Discrete Mathematics and Applications, P. Toth and D. Vigo, Eds. Philadelphia: SIAM.

Towards a Napster-Like P2P B&B algorithm

M. Mehdi, M. Mezmaz, N. Melab and E-G. Talbi
Laboratoire d'Informatique Fondamentale de Lille
UMR CNRS 8022, INRIA Futurs - DOLPHIN Project
Cité scientifique - 59655, Villeneuve d'Ascq cedex - France
{mehdi,mezmaz,melab,talbi}@lifl.fr

Abstract— **The Branch and Bound (B&B) algorithm is one of the most used methods to solve in an exact way combinatorial optimization problems. In a previous article, we proposed a new approach of the parallel B&B algorithm for distributed systems. This approach is based on a new way to efficiently deal with some crucial issues met in distributed systems. The new method is used to propose a parallilization of the B&B with the farmer-worker paradigm. The obtained resluts show the efficiency and the scalability of the approach.**

However, the new farmer-worker approach has a disadvantage: some nodes of the B&B tree can be explored by several B&B processes. To avoid this redundant work, we propose a new approach based on the Napster-like Peer-to-Peer(P2P) model. Validation is performed by experimenting the approach on a bi-objective flow-shop problem instance that has never been solved exactly. The obtained results, after 15 days on computation pool of about 2500 processors, belonging to 8 distinct clusters, prove the efficiency of the proposed approach. Indeed, the peer processors were exploited on average to 99.3% while the index processor was exploited 0.01%.

Keywords— **Branch and Bound, Parallel Computing, Peer-to-Peer Computing, Flow-Shop Problem.**

I. Introduction

Combinatorial optimization addresses problems for which the resolution consists in finding the (near-)optimal configuration(s) among a large finite set of possible configurations. In practice, most of these problems are naturally NP-hard and complex. The Branch and Bound (B&B) algorithm is one of the most popular methods to solve exactly this kind of problems. This algorithm allows to reduce considerably the computation time required to explore all the solution space associated with the problem being solved. However, the exploration time remains considerable, and using parallel processing is one of the major and popular ways to reduce it. Many parallel B&B approaches have been proposed in the literature. A taxonomy of associated parallel models is presented in [12]. Four models are mainly identified and studied within the context of distributed computing. The parallel exploration of the search tree is one the most used one.

[13] proposes a farmer-worker approach based on special coding of the explored tree and the work units. These codings allow to optimize the dynamic distribution and check-pointing mechanisms on distributed systems, to implicitly detect the termination of the algorithm, and to efficiently share the global information. The algorithm has been applied to the flow-shop scheduling problem, one of the hardest challenging problems in combinatorial optimization. Using the new approach, the problem instance (50 jobs on 20 machines) has been optimally solved for the first time. The method allows not only to improve the best known solution for the problem instance but it also provides a proof of the optimality of the provided solution.

However, the new farmer-worker approach has a disadvantage: some nodes of the tree can be explored by several B&B processes. To avoid this redundant work, it is indispensable for the B&B processes to communicate and cooperate during the resolution. The approach thus must be deployed according to the (Peer-to-Peer)P2P paradigm. The strategies of Napster [9] and Gnutella [11] are the two main approaches used in P2P systems. Unlike the approach of Gnutella, the model of Napster requires minor modifications to adapt the farmer-worker paradigm and to get a P2P deployment without a redundancy in the tree exploration. To the best of our knowledge, the presented approach in this paper is the first use of the Napster-like P2P model for computing systems. The goal of the new approach is to be as efficient as the farmer-worker approach, to avoid the redundancy in work during the exploration of the B&B tree, and to prove that Napster-like P2P model is efficient for computing systems. Unlike the farmer-worker approach, the Napster-like P2P approach can be compared easily with the sequential B&B since no node of the tree is visited more than one time.

The rest of the paper is organized as follows. **Section II** and **Section III** give an overview of the B&B algorithm, its parallelization, and related works. The parallel approach proposed in [13] is described in **Section III**. **Section V** presents its implementation based on the farmer-worker paradigm. In **Section VI**, we describe our new Napster-like P2P approach for the parallelization of the B&B algorithm. **Section VII** presents the experiment performed on a bi-objective flow-shop instance to evaluate the quality of the approach. **Section VIII** draws some conclusions and perspectives of this work.

II. Parallel B&B Algorithm

Solving a problem in combinatorial optimization consists in exploring a search space to provide a (near-)optimal solution. To each candidate solution of the

This work is part of the *CHallenge in Combinatorial Optimization (CHOC)* project supported by the *National French Research Agency (ANR)* through the *Hign-Performance Computing and Computational Grids (CIGC)* programme.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

search space is associated a cost. Solving exactly a combinatorial optimization problem consists in finding the solution having the optimal cost. For this purpose, the B&B algorithm is based on an implicit enumeration of all the solutions of the considered problem. The search space is explored by dynamically building a tree whose root node represents the problem being solved and its whole associated search space. The leaf nodes are the potential solutions and the internal nodes are subspaces of the total solution space. The size of these subspaces is increasingly reduced as one approaches the leaves.

The construction of such a tree and its exploration are performed using four operators: *branching*, *bounding*, *selection* and *elimination*. The algorithm proceeds in several iterations during which the best solution found so far is progressively improved. The generated and not yet treated nodes are kept in a list whose initial content is limited to only the root node. The four operators intervene at each iteration of the algorithm. The B&B makes it possible to reduce considerably the computation time necessary to explore the whole solution space. However, this remains considerable and parallel processing is thus required to reduce the exploration time.

In [12], four parallel models are identified for B&B algorithms: (1) *the parallel multi-parametric model*, (2) *the parallel tree exploration*, (3) *the parallel evaluation of the bounds*, and (4) *the parallel evaluation of a single bound*. The model (1) consists in launching simultaneously several B&B processes. These processes differ by one or more operators, or have the same operators, but parameterized differently. The trees explored in this model are not necessarily the same. Model (1) guarantees the implicit exploration of the whole solution space. Like model (1), model (2) also consists in launching several B&B processes. However, all the processes in model (2) are similar, and explore simultaneously the same tree. Among the four models, this model is the most popular and studied one. Unlike the two previous models, models (3) and (4) suppose the launching of only one B&B process. They do not allow to parallelize the whole B&B algorithm as both models (1) and (2) do, but they parallelize only the bounding operator. In model (3), each process evaluates the bounds of a distinct pool of nodes, while in model (4) a set of processes evaluate in parallel the bound of a single node.

In [12], an analysis of these different parallel models is presented within the context of distributed computing systems. Distributed systems in general, and P2P systems in particular, exploit the resources of a great number of machines. These resources can be processors, memories or others. A distributed system aims at giving the illusion of a very powerful virtual machine. It makes it possible to solve problems which require very long execution time. Since distributed systems are distributed memory systems, the parallel tree exploration model is more suitable for these environments.

III. Related Works

Many parallel B&Bs on distributed computing systems are described in the literature. [12], [5] and [15] present different parallel strategies for the B&B algorithm. On the distributed systems, the B&B algorithm is often deployed according to the master-slave paradigm. The master manages the list of the not yet explored nodes and distributes nodes to the slave machines. A slave machine receives one node only from the master, explores the subtree of which the received node is the root, and returns to the master all not explored nodes. From a deployment to another, what change often is the condition of returning these nodes. In [3], a slave machine returns all the not explored nodes after a hundred seconds. In [1], a slave machine explores only the son nodes of the received node, and returns the result to the master.
The best parallel efficiency recorded by [3], on a platform of 185 processors, is equal to 85.6%. This result is obtained by exploiting on average only 17 processors during this test. While the best parallel efficiency recorded by [1], on a grid of 128 processors, is equal to 71%. Besides, [2] shows the limits of this paradigm and the used load balancing strategy. [2] advises to use to the hierarchical master-slave paradigm. However, by using 348 processors organized with hierarchical master-slave paradigm, the best parallel efficiency obtained by [2] is about 33%.
[8] proposes an original P2P strategy. These strategy is often referenced in the combinatorial optimization literature on distributed systems. However, the obtained parallel efficiency are less of the one obtained in [3]. Indeed, [8] obtains 84.4% on a simulator of 100 processors.

IV. Fold-Unfold Approach

The proposed approach in [13] is based on the *parallel tree exploration model* with a depth first *search strategy*. This approach is focused on the list of active nodes. The B&B active nodes are those generated but not yet treated. During a resolution, this list evolves constantly and the algorithm stops once it becomes empty. Any list of active nodes covers a set of tree nodes. This set is made up by all nodes which can be explored from a node of this active list. The principle of the approach is based on the assignment of a number to each node of the tree. The numbers of any set of nodes, covered by a list of active nodes, always form an interval. The approach thus defines a relation of equivalence between the concept of *list of active nodes* and the concept of *interval*. So, it is possible to deduce a list of active nodes from an interval, and an interval from a list of active nodes. As its size is reduced, the interval is used for communications and check-pointing, while the list of active nodes is used for exploration.
In order to pass from one concept to the other, the approach defines two additional operators: the *fold opera-*

tor and the *unfold operator*. The fold operator deduces an interval from a list of active nodes, and the unfold operator deduces a list of active nodes from an interval.

V. Farmer-Worker Parallel Approach

Fold and Unfold operators can be used for the parallelization of the B&B according to different parallel paradigms. In [13], the selected paradigm is the farmer-worker one. In this paradigm, only one host plays the role of the farmer, and all the other hosts play the role of a worker. This paradigm is relatively simple to be used. Its major disadvantage is that the farmer can constitute a bottleneck. However, communicating and handling intervals instead of list of active nodes make it possible to reduce the communication costs and the farmer work. This paradigm is thus selected in [13] to test the fold-unfold approach. The goal is to show that the approach makes it possible to push the limit of this paradigm as for the bottleneck, and to have thus a more scalable approach.

In the adopted farmer-worker approach, the workers host as many B&B processes as they have processors, and the farmer hosts the coordinator. Each B&B process explores an interval of node numbers, and manages local solution set. To get a work, a B&B process obtains an interval from the coordinator, deduces a list of nodes from this interval using the unfold operator, and explores this node list. To do a check-point, a B&B process deduces an interval from the list of not treated nodes using the Fold operator, and communicates this interval to the coordinator. The cost of the fold and unfold operators are infinitely negligible compared to the time devoted to exploring the B&B tree. On the other hand, the coordinator keeps a copy of all the not yet explored intervals, and manages global solution set. The copies of the intervals are kept in a set noted $INTERVALS$, and the solutions of the global solutions in another set noted $SOLUTIONS$. Fig. 1 gives an example with three B&B processes and a coordinator. In this example, three intervals are being explored, and the fourth one is waiting for a free available B&B process.

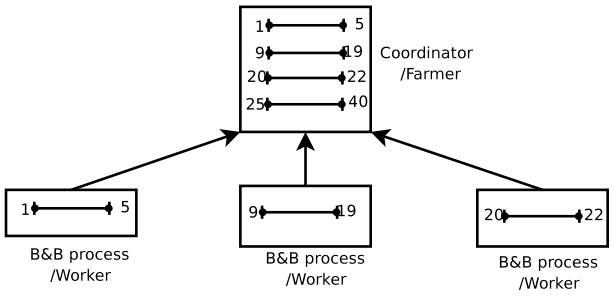


Fig. 1. An example with B&B processes and a coordinator

In addition to balancing the load between B&B processes, other problems must be taken into account. Indeed, the B&B processes make two assumptions about the workers. They suppose that they are likely to break down and not necessary dedicated. Consequently, these processes are fault tolerant and are launched according to the cycle stealing model. The only assumption of the coordinator about the farmer is that it can fail. The coordinator manages only the possible failures of the farmer.

The approach presented in [13] has been implemented following a large scale idle time stealing paradigm (Farmer-Worker). It has been experimented on a flow-shop problem instance ($Ta056$) that has never been optimally solved. The new algorithm allowed to realize a success story as the optimal solution has been found with proof of optimality, within 25 days using about 1900 processors belonging to 9 Nation-wide distinct clusters (administration domains). During the resolution, the worker processors were exploited with an average of 97% while the farmer processor was exploited only 1.7% of the time. These two rates are good indicators on the efficiency of this approach and its scalability.

VI. Napster-Like P2P Approach

However, the presented farmer-worker approach in [13] has a disadvantage: some nodes of the tree can be explored by several B&B processes. Let [A,B[an interval being explored by a holder B&B process, and [A',B'[its copy in the coordinator. As explained in [13], the interval [A',B'[can be divided into two intervals [A',C[and [C,B'[. This occurs after a request from requesting B&B process. After this division, [C,B'[is explored by the requesting process, while the holder process continues the exploration of [A,B[. However, [C,B'[and [A,B[are not completely disjoint. Consequently, the same nodes of the tree can be explored by the two processes.

To avoid this redundant treatment, it is indispensable for requesting and holder processes to coordinate their intervals. Before beginning the exploration of the received interval, the requesting process must make sure that the two intervals are disjoint. It is essential to requesting process to contact the process holder in order to refine the division. The holder process is then given the responsibility to determine the intervals that each of the two processes must explore. This is done by the intersection and subtraction operators. These two operators are noted $\cap$ and $\setminus$, respectively. The equations (1) and (2) define them.

$$[A,B[\cap[C,D[\quad = \quad [max(A,C),min(B,D)[\tag{1}$$

$$[A,B[\setminus[C,D[\quad = \quad \{X/X \in [A,B[\ \ and \ \ X \notin [C,D[\} \tag{2}$$

Let [A,B[the interval of the holder process, and [C,D[the received interval by the requesting process. The holder process proceeds in two stages. First, it subtracts from [C,D[the already explored interval by the

holder process. Then, it subtracts [A,B[from the given interval to the requesting process. In other words, $[C, D[$ and $[A, B[$ become equal to $[C, D[\cap[A, B[$ and $[A, B[\backslash[C, D[$, respectively. Once this new division finished, both intervals obtained are completely disjoint, and both processes can continue the exploration of their interval. This new strategy thus ensures that no nodes is explored twice.

Unlike the previous approach, this new approach supposes that a B&B process can be contacted by another B&B process. Indeed, the B&B processes send requests to the coordinator and to the other B&Bs processes, and can receive requests from any B&B process. The approach thus must be deployed according to the P2P paradigm. In this new approach, a peer hosts as many B&B processes it has processors. The intervals constitute the handled resources. In a P2P deployment, one important issue must be taken into account: the resource discovery and their routing.

The resource discovery consists in identifying the peer able to provide the required resource. As indicated in [4], the strategies of Napster [9] and Gnutella [11] are the two main approaches used in P2P systems.

In Naspter, the resource discovery is centralized, and based on an index peer which stores the peer-resource relations. When a peer seeks a resource, it obtains from the index the address of the peer having this resource.

Unlike the centralized strategy of Napster, the resource discovery in Gnutella is completely distributed. In Gnutella, a peer is connected only by its neighbors. They are logical neighbors and the set of connections forms logical topologies. To find a resource, a peer broadcast a message to its neighbors in this logical topology. The request is propagated by a neighbor to its own neighbors. A peer, which receives the request and which has the required resource, does not propagate the message, and returns the resource to the requesting peer.

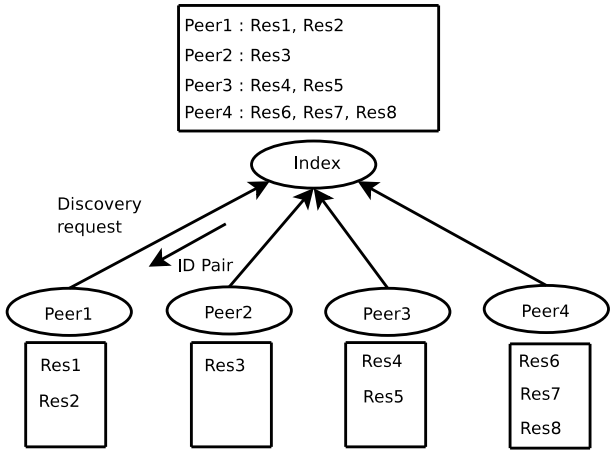


Fig. 2. Resource discovery in Napster

To deploy the B&B, our approach is based on the resource discovery model of Napster. Unlike the approach of Gnutella, the model of Napster requires minor modifications to adapt the farmer-worker paradigm, and to get a P2P deployment without a redundancy in the tree exploration. The coordinator process plays the role of index, and the worker processes the role of the peers. In the farmer-worker paradigm, the coordinator returns to a worker an interval. In this new strategy, the index returns to a peer the resource and the address of the peer which holds this resource. As already explained, this resource is an interval. Then, the requesting peer contacts the holder peer of the received interval in order to make disjoint the intervals of the holder and requesting peers.

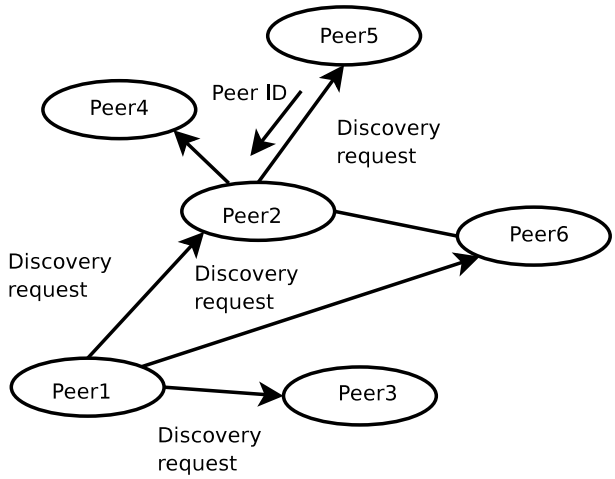


Fig. 3. Resource discovery in Gnutella

VII. Experimentation

A. Bi-objective permutation ow-Shop problem

The flow-shop problem is one of the numerous scheduling multi-objective problems [14] that has received a great attention given its importance in many industrial areas. The problem can be formulated as a set of N jobs $J_1, J_2, \ldots, J_N$ to be scheduled on M machines. The machines are critical resources as each machine can not be simultaneously assigned to two jobs. Each job J_i is composed of M consecutive tasks $t_{i1}, \ldots, t_{iM}$, where t_{ij} designates the j^{th} task of the job J_i requiring the machine m_j. To each task t_{ij} is associated a processing time p_{ij}, and each job J_i must be achieved before a due date d_i.

The problem being tackled here is the bi-objective permutation flow-shop problem where jobs must be scheduled in the same order on all the machines. Therefore, two objectives have to be minimized: (1) C_{max}: makespan (Total completion time), (2) T: total tardiness. The task t_{ij} being scheduled at time s_{ij}, the two objectives are NP-hard[7][10], and can be formulated as follows:

$$C_{max} = Max\{s_{iM} + p_{iM} | i \in [1 \ldots N]\}$$

$$T = \sum_{i=1}^{N} [max(0, s_{iM} + p_{iM} - d_i)]$$

The application of the proposed approach to the flow-hop problem has been experimented on one of the instances proposed by [6]. More exactly, it is an instance generated for problems of 50 jobs on 5 machines in

which only the makespan[1] is considered. The instance has been extended with the tardiness[2] as the second objective. Such instance has never been solved exactly in its bi-objective formulation.

B. The experimentation platform

The method is tested on the computational pool detailed in Table I. It is made up of approximately $2,500$ processors belonging to 8 clusters. These clusters belong to Grid'5000[3]. Grid'5000 is a Nation-wide experimental grid composed by 9 clusters distributed over several French universities (Bordeaux, Grenoble, Lille, Nancy, Orsay, Rennes, Sophia, Toulouse). The eight exploited clusters are those of Bordeaux, Lille, Nancy, Orsay, Rennes, Sophia, Toulouse. All the machines of Grid'5000 are dedicated bi-processors, and interconnected by the Ethernet Gigabit, using RENATER[4] nation-wide network.

CPU model	Cluster	Number of CPU
AMD 2.2	Bordeaux	2x58
Xeon 3.0		2x43
AMD 2.2	Lille	2x53
AMD 2.6		2x46
AMD 2.0	Lyon	2x56
AMD 2.4		2x70
AMD 2.0	Nancy	2x47
Xeon 1.6		2x120
AMD 2.0	Orsay	2x216
AMD 2.4		2x126
AMD 2.2	Rennes	2x64
AMD 2.0		2x100
AMD 2.0		2x74
AMD 2.2	Sophia	2x56
AMD 2.6		2x50
AMD 2.2	Toulouse	2x58
Total		**2,474**

TABLE I: The computational pool

C. Experimental results

After about 15 days of computation, the experiment did not make it possible to solve the instance. However, the recorded statistics seem enough to evaluate the quality of the approach. Table II summarizes the most important statistics recorded during the resolution. The experiment lasted approximately 15 days, with an average of 655 processors, a maximum of $1,606$ available processors, and a cumulative computation time of about 26 years. About 7 billion nodes were explored.

As Table II indicates, more than 4 million checkpoint operations were done by the B&B processes, while the index did its check-point about 7 hundred times. These check-point operations have allowed the fault tolerance mechanism to face the thousands of failures which have occurred. Indeed, more than 150 thousand failures of the peers and 31 failures of the index were recorded.

The peer processors were exploited on average to 99.3% while the index processor was exploited 0.01%. These two rates are good indicators on the parallel efficiency of this approach and its scalability. In a Napster-like P2P paradigm, a good approach must maximize the exploitation rate of the peer processors and must minimize the exploitation rate of the index processor.

Running wall clock time	15 days
Total CPU time	26 years
Average number of peers	655
Maximum number of peers	1,606
Number of explored nodes	6,782,787,073
Index check-point operations	698
Peer check-point operations	4,581,950
Peer failures	169,141
Index failures	31
Peer CPU exploitation	99.3%
Index CPU exploitation	0.01%

TABLE II: The computation statistics

VIII. Conclusions and Future Works

Solving exactly large instances of combinatorial optimization problems requires a huge amount of computational resources. Parallel Branch and Bound(B&B) algorithms based on the parallel exploration of the search tree have successfully been applied to solve these problems. However, experiments are often limited to few tens of processors. Designing and implementing B&B algorithms for a large scale computational distributed systems is a great research challenge as several crucial issues must be tackled. In [13], we have proposed a new B&B algorithm with new approaches allowing to efficiently tackle the problems met in distributed systems. The approach consists in an efficient coding associated with the explored tree and work units (collections of nodes). The approach has been implemented following a large scale idle time stealing farmer-worker paradigm. The obtained resluts show the efficiency and the scalability of the approach.

However, the new farmer-worker approach has a disadvantage: some nodes of the B&B tree can be explored by several B&B processes. To avoid this redundant work, we propose a new approach based on the Napster-like Peer-to-Peer(P2P) model. Validation is performed by experimenting the approach on a bi-objective flowshop problem instance that has never been solved exactly. The obtained results, after 15 days on a computation pool of about 2500 processors, belonging to 8 distinct clusters, prove the efficiency of the proposed approach. Indeed, the peer processors were exploited on average to 99.3% while the index processor was exploited 0.01%. These two rates are good indicators on

[1] http://www.eivd.ch/ina/Collaborateurs/etd/default.htm
[2] http://www.lifl.fr/OPAC/
[3] http://www.grid5000.fr
[4] http://www.renater.fr

the parallel efficiency of this approach and its scalability. To the best of our knowledge, the presented approach is the first use of the Napster-like P2P model for computing systems. The new approach is as efficient as the farmer-worker approach, avoid the redundancy in work during the exploration of the B&B tree, and prove that Napster-like P2P model is efficient for computing systems.

Unlike the farmer-worker approach, the Napster-like P2P approach can be compared easily with the sequential B&B since no node of the tree is visited more than one time. We plan to study the variation of efficiency according to the number of peers. It is also planned to use the approach with an other P2P paradigm to push far the scalability limits of the Napster-like P2P model. The objective is to exploit more and more processors and to solve more and more complex instances.

Acknowledgment

We would like to thank the *National French Research Agency* (ANR). This work is part of the *CHallenge in Combinatorial Optimization (CHOC)* project supported by the ANR through the *High-Performance Computing and Computational Grids (CIGC)* program. Our thanks are also addressed to the persons in charge, to the engineers and to the technicians of the Grid'5000 clusters.

References

[1] K. Aida and Y. Futakata. High-performance parallel and distributed computing for the BMIeigenvalue problem. *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 71–78, 2002.

[2] K. Aida and T. Osumi. A case study in running a parallel branch and bound application on the grid. *Applications and the Internet, 2005. Proceedings. The 2005 Symposium on*, pages 164–173.

[3] K. Anstreicher, N. Brixius, J.P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588, 2002.

[4] Franck Cappello. Calcul Global Pair a Pair : extension des systemes Pair a Pair au calcul. *Lettre de l'IDRIS*, pages 14–25, month=Janvier,, 2002.

[5] D.Gelenter and T.G.Crainic. Parallel Branch and Bound Algorithms: Survey and Synthesis. *Operation Research*, pages 42:1042–1066, 1994.

[6] E.Taillard. Banchmarks for basic scheduling problems. *European Journal of European Research*, pages 23:661–673, 1993.

[7] Johnson D.S. Garey M.R. and Sethi R. The complexity of flow-shop and job-shop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[8] A. Iamnitchi and I. Foster. A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems. *29th International Conference on Parallel Processing (ICPP), Toronto, Canada, August*, pages 21–24, 2000.

[9] N. Inc. The Napster homepage. *Online: http://www. napster. com*, 2000.

[10] Du J. and Leung J. Y.-T. Minimizing Total Tardiness on One Machine is NP-hard. *Mathematics of operations research*, 15:483–495, 1990.

[11] G. Kan. Gnutella. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, 2001.

[12] N. Melab. *Contributions à la résolution de problèmes d'optimisation combinatoire sur grilles de calcul.* PhD thesis, LIFL, USTL, Novembre 2005.

[13] M. Mezmaz, N. Melab, and E-G. Talbi. A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems. In *In Proc. of 21th IEEE Intl. Parallel and Distributed Processing Symp.*, Long Beach, California, March 2007.

[14] V. T'kindt and J-C. Billaut. *Multicriteria Scheduling - Theory, Models and Algorithms.* Springer-Verlag, 2002.

[15] H. Trienekens and A. de Bruin. Towards a taxonomy of parallel branch and bound algorithms. Report EUR-CS-92-01, Department of Computer Science,Erasmus University Rotterdam, 1992.

Distributed Coevolutionary Genetic Algorithm for Optimal Design of Ad Hoc Injection Networks

Grégoire Danoy, Pascal Bouvry
Faculty of Sciences, Technology and Communications
University of Luxembourg
Luxembourg
{gregoire.danoy, pascal.bouvry}@uni.lu

Enrique Alba
Department of Computer Science
University of Málaga
Málaga, Spain
eat@lcc.uma.es

Abstract— **Multi-hop ad-hoc networks allow establishing local groups of communicating devices in a self-organizing way. However, in a global setting such networks fail to work properly due to network partitioning. This means that users locally interacting could eventually spread and move away from each other and consequently loose their connections. Considering that devices are capable of communicating both locally (e.g. using Wi-Fi or Bluetooth) and additionally with remote devices (e.g. using GSM/UMTS links) the objective of our work is to optimize the way of inter-linking multiple network partitions. To this end we rely on small-world network properties, that consist in using special attributes like the clustering coefficient and the characteristic path length. In this paper we investigate the use of a distributed Cooperative Coevolutionary Genetic Algorithm (CCGA) and compare its performance to a generational and a steady state genetic algorithm (genGa and ssGA) for optimizing one instance of this topology control problem and present initial evidence of its capacity to solve it.**

I. Introduction

Multi-hop ad-hoc networks are networks composed of communicating devices capable of spontaneously interconnecting without any pre-existing infrastructure. The most popular wireless networking technologies available nowadays for building such networks are Bluetooth and IEEE802.11 (Wi-Fi). Devices in range to one another communicate in a point-to-point fashion. But such ad-hoc networks are intrinsically dynamic. Due to their limited transmission range, such networks face partitioning problems that penalizes their global efficiency. In real scenarios, one or more additional remote links have to be created to keep connected the different clusters of locally interacting users that dynamically move.

In this paper we consider the problem of optimizing the addition of such long-range links (e.g. GSM, UMTS or HSDPA) that are also called *bypass links* to interlink network partitions. To tackle this topology control problem, we use small-world properties as indicators for the good set of rules to maximize inter-link efficiency. Small-world networks [1] feature a high clustering coefficient (γ) while still retaining a small characteristic path length (L). A small path length corresponds to fewer hops, which is of importance for effective routing mechanisms as well as for the overall communication performance of the entire network. The clustering coefficient represents the connectivity in the neighborhood of each node and thus reflects the degree of information dissemination each single node can achieve. This finally motivates the objective of evoking small-world properties in such settings.

In order to optimize those parameters (maximizing γ, minimizing L) and to minimize the number of required bypass links in the network, we relied on Evolutionary Algorithms (EAs) [2] and more specifically on a distributed Cooperative Coevolutionary Genetic Algorithms (GA) [3] using Dafo, our distributed agent framework for evolutionary optimization. CCGA has already proved its ability for solving complex real-world problems [4]. We start by investigating the kind of evolution step more amenable to our problem by comparing the performance of a distributed CCGA, to both generational [5] and steady-state [6] GAs on a basic instance of a partitioned ad-hoc network.

The remainder of this paper is organized as follows. In the next section we give a detailed view on CCGA. Section III introduces Dafo, our distributed agent framework for evolutionary optimization. Then in Section IV, we provide details on the injection network problem; we address as well several small-world properties. In Section V presents the experiments and analyzes the results. The last section contains our conclusions and perspectives.

II. Coevolutionary algorithms

As for the "classical" genetic algorithm, the concept of coevolutionary algorithms comes from the biological observations [7]. Indeed, the nature is composed of several species that coevolve. And instead of evolving a population of similar individuals (like in classical Genetic Algorithms) representing a global solution, we consider the coevolution of subpopulations of individuals representing specific parts of the global solution. In the following subsections, we introduce CCGA, a Cooperative Coevolutionary Genetic Algorithms [3]. This algorithm was already applied (cf. [8]) for parallel and distributed optimization of a number of test functions known in the area of evolutionary computation. It was demonstrated that coevolutionary algorithms outperform a sequential GA. In the present article, we consider optimizing the Injection Networks problem by applying this cooperative coevolutionary algorithm.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

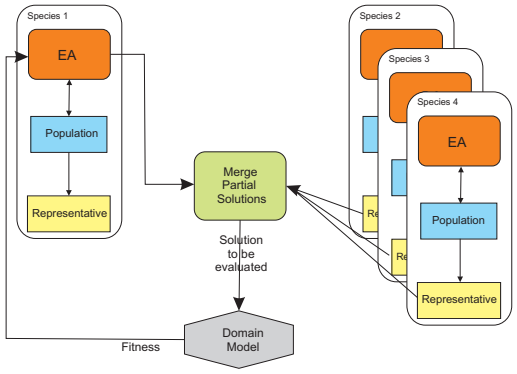


Fig. 1. Potter and De Jong's CCGA architecture

Algorithm 1: CCGA

```
gen = 0
foreach species_s do
    Pop_s(gen) = randomly initialized population
    evaluate fitness of each individual in Pop_s(gen)
end
while termination condition = false do
    gen = gen + 1
    foreach species_s do
        select Pop_s(gen) from Pop_s(gen − 1) based on
        fitness
        apply genetic operators to Pop_s(gen)
        evaluate fitness of each individual in Pop_s(gen)
    end
end
```

A. CCGA

Cooperative (also called symbiotic) coevolutionary genetic algorithms (CCGA) involve a number of independently evolving species which together form complex structures, well-suited to solve a problem. The fitness of an individual depends on its ability to collaborate with individuals from other species. In this way, the evolutionary pressure stemming from the difficulty of the problem favors the development of cooperative strategies and individuals. Potter and DeJong [3] developed a model in which a number of populations explore different decompositions of the problem. In Potter's system, each species represents a subcomponent of a potential solution. Complete solutions are obtained by assembling representative members of each of the species (populations). The fitness of each individual depends on the quality of (some of) the complete solutions it participated in, thus measuring how well it cooperates to solve the problem. The evolution of each species is controlled by a separate, independent evolutionary algorithm. In the initial generation *(t=0)* individuals from a given subpopulation are matched with randomly chosen individuals from all other subpopulations. A fitness for each individual is evaluated, and the best individual in each subpopulation is found. The process of *cooperative coevolution* starts form the next generation *(t=1)*. For this purpose, in each generation a cycle of operations is repeated in a round-robin fashion. Only one current subpopulation is active in a cycle, while the other subpopulations are frozen. All individuals from the active subpopulation are matched with the best values of frozen subpopulations. When the evolutionary process is completed a composition of the best individuals from each subpopulation represents a solution of a problem. Figure 1 shows the general architecture of Potter's cooperative coevolutionary framework, and the way each evolutionary algorithm computes the fitness of its individuals by combining them with selected representatives from the other species. Potter's methods have also been used or extended by other researchers, for instance Eriksson and Olsson [4] have used a cooperative coevolutionary algorithm for inventory control parameter optimization.

III. Distributed Agent-based evolutionary computation

We consider the opportunity to embed our players into software agents what is a convenient and elegant way to benefit from existing software infrastructure. Indeed using a multi-agent framework like Madkit [9] leverages us of writing low-level agent interaction behaviors and highly simplifies the agents distribution.

Since its introduction in the 70's, the agent technology has become synonymous with advanced computer software. The nature of real world problems has lead to the evolution of multi-agent systems in Distributed Artificial Intelligence (DAI). In this case each aspect of a problem is under the control of an agent and all the agents in the system interact to generate a global solution. In the proposed approach, the agent environment is composed of other evolving agents. In this article we choose Evolutionary Algorithms for modelling agents intelligence and the concept of agents organization for modelling agents interactions.

The concept of a "computational agent" becomes increasingly important in computer science, representing a new level of abstraction for software design. Distributed artificial intelligence/multi-agent systems are typically applied in two ways. In the first, the problem domain is itself distributed, e.g. telecommunications routing, and as such the multi-agent paradigm is a natural "fit" since each aspect of the system can be attributed to an agent. In the second, complex tasks are divided into multi-aspect problems to allow for the construction of a solution through the combination of a number of simpler (in respect to the global task) interacting agents. New issues arise when evolutionary computation is applied to the multi-agent paradigm. In these systems evolutionary algorithms must adapt to dynamic problem spaces, where changes are caused by the interactions of the agents in the environment of the global system. Agents evolve in a dynamic environment composed of other agents.
The use of evolutionary computing techniques in systems containing many interacting agents/entities goes back to the earliest days of experiments in machine intelligence. For example, Barricelli [10] used an abstract ecological model to examine the evolution of complexes

of cooperative entities, based in the idea of "symbiogenesis", the evolution of complexity by the bringing together of previously autonomous entities. However, nowadays most of coevolutionary computing consists of systems in which agents roles are predetermined as being either competitive or cooperative or a mixture of the two, i.e. agents are assigned particular tasks within the global system. Distributed problem solving by a multi-agent system represents a promising approach for solving complex computational problems. An agent-oriented problem-solving environment increases efficiency, capability and genericity by employing a set of agents communicating and co-operating to achieve their goals, i.e. that is to find local solutions that satisfy both their hard and soft constraints.

Our solution consists in providing a meta-level in the form of a distributed agent framework dedicated to evolutionary optimization including coevolutionary genetic algorithms. Modelling the algorithm with a multi-agent system makes explicit resolution strategy (i.e. the algorithm interaction graph) by using organizational models explicitly representing the roles and the interactions that are allowed for each agent. Using the agent technology also allows us to take benefit from existing multi-agent platforms and methodologies. The deployment and the distribution of the algorithm is thus ensured.

A. Framework Architecture

By providing a minimum of Java code concerning his optimization problem and a simple XML configuration file, the designer is capable of optimizing its function using various GAs that are generational GA, steady state GA and CCGA (that can be distributed). The XML file is used as an input file by the *Organizer Agent* for specifying information about the genetic algorithm, its parameters and information concerning the distribution if required.

Figure 2 represents a simple example of a distributed instance of Dafo on 3 different computers. Computers 2 and 3 run a *Slave Scheduler* Agent that first sends message *1* (which contains the IP address of Computer 1) to the *Communicator* Agent running in the Madkit platform's kernel in order to connect to Computer 1 (represented by message *2*). As soon as all *Communicators* have established a connection with the *Communicator* of Computer 1, one agent can communicate with any other agent, no matter on which computer it is (i.e. the computers are fully connected). Consequently, after this connection, all the other agents can communicate in a fully transparent way. Once all nodes are connected, the *Master Scheduler* sends message *3* that contains parameters (topology, population size, crossover operator, etc.) that will be used to instantiate the *Evolutionary Agents* (running a Simple GA), as represented by message *4*. The *Evolutionary Agents* can freely communicate with each other according to the CCGA architecture (as presented in Figure 1 and will send their partial solutions (message *4*) to the *Observer Agent* that is in charge of merging those

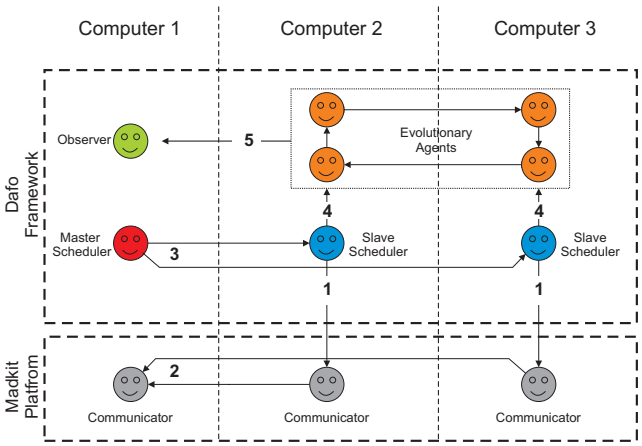


Fig. 2. Distributed Dafo Architecture

results and provide as output the global solution found.

Using Dafo makes juggling with the different versions of GAs easy. For instance, to use CCGA in distributed mode, it is sufficient to add the IP address and the port of the Node 1 and the total number of nodes in the configuration file.

IV. The Problem

This section introduces the injection network optimization problem using small-world properties. We first provide the reader with a definition of the injection network concept. Next we give details on what defines a small-world graph in this context.

A. Injection Networks

Due to several difficult (and practical) challenges that are inherent to mobile multi-hop ad-hoc networks, some past work advises the utilization of hybrid wireless networks, where a fixed infrastructure supports a higher connectivity among several clusters of ad hoc networks and avoids network partitioning [11] [12] [13]. However such a hybrid wireless network is often not feasible, because of economical and implementation issues. Alternatively, an infrastructureless setting is of interest where problems of restricted geographical regions are avoided. Helmy [14] focuses on long-range links for which the objective is to reduce the number of queries during the search for a given target node. Another approach introduces base stations to increase connectivity in ad hoc networks [15], thus realizing global reachability. Watts [1] introduces a spatially defined link, called *global edge*, with length-scaling properties to include spatial models in his investigations. Some approaches extend standard ad-hoc network models, by considering two different transmission ranges [16] [17], e.g. small distance Bluetooth links along with higher distance Wi-Fi links. Our initial motivation for the current investigation is based on the assumption that technologies like Bluetooth and Wi-Fi can be used to create ad-hoc communication links within the transmission range at no charge. Additional cellular network links such as GSM/UMTS/HSDPA might be employed by appropriately equipped devices to establish supple-

mentary communication links between two arbitrary devices. These links, however, will induce additional costs. Furthermore, we propose to implement that in a transparent way for the end user, i.e. linking in a mobile multi-hop ad-hoc network should be managed without explicit human interaction (self-organization). In summary, different approaches exist to augment ad-hoc networks with additional links. Different reasons exist for this need on increasing connectivity: e.g. to gain bandwidth between particular devices, and also to inter-link multiple ad-hoc network partitions. Consequently, we introduce the notion of *bypass links*.

Definition 1 (Watts) The spatial neighborhood $\Gamma_{tr}(v)$ of a node v is the set of nodes within transmission range tr of v.

Definition 2: A bypass link is a link *(u,v)* between nodes u and v with $u \notin \Gamma_{tr}(v)$.

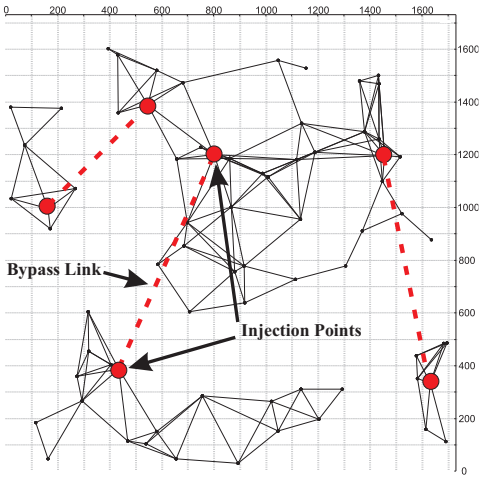


Fig. 3. Example of Injection Network

That is, a bypass link is a link which connects two nodes that are not in the same spatial neighborhood. Please note that elements of $\Gamma_{tr}(v)$ do not necessarily have to be connected to v in real settings. Practically, a bypass link can be built by using a cellular network as well as by using access points. Nevertheless, in our model a bypass link is counted as a single hop, thus simplifying the real topology behind that bypass link. Since we can dynamically control such bypass links, the network topology basically can be biased as needed, resulting in a topology control problem. Thus, this approach does not need to consider mobility models as mobility can be compensated via the bypass links. The injection communication paradigm is based on establishing bypass links between carefully selected devices. Herrmann et al. [18] called these dedicated communication points *hub nodes*. Depending on the overall objective, the selection of such devices can be driven by different factors, like for instance to obtain a high local clustering coefficient around the hub, or devices showing certain attributes (e.g. information available or services offered on a particular device). These dedicated devices used for establishing bypass links are called *injection points*. For self-organizing communication networks based on bypass links and injections points as described before we use the term *injection networks*.

Definition 3: Two nodes u and v are called injection points if a bypass link *(u,v)* exists between nodes u and v.

In order to study the small-world properties of such hybrid networks, we had to rely on some ad-hoc network simulator. In our case we used Madhoc [19], an application-level network simulator dedicated to the simulation of mobile ad hoc networks. The main motivation for using Madhoc is its ability to simulate hybrid networks, i.e. mixing different technologies (e.g. bluetooth/Wi-Fi for local connections and UMTS for long distance calls), and its graphical and batch modes of visualization.

B. Small-Worlds

Small-world networks [1] are a class of random graphs that exhibit two main characteristics: a small characteristic path length (L) and a high clustering coefficient (γ). A formal definition of these two graph measures is given below:

Definition 4 (Watts) The local clustering coefficient γ of one node v with k_v neighbors is

$$\gamma_v = \frac{|E(\Gamma_v^r)|}{\binom{k_v}{2}}$$

where $|E(\Gamma_v^r)|$ is the number of links in the relational neighborhood of v and $\binom{k_v}{2}$ is the number of possible links. The clustering coefficient is the average local clustering coefficient for all nodes of a network.

For example, in Figure 4, node a is connected to three nodes b, d and e. The maximum number of possible edges among these three nodes is three. The graph shows that only two out of those three possible edges exist (between b-e and d-e). The edge b-d is missing. So the clustering coefficient for node a is 2/3 or about 0.67. For Figure 4, this value is 0.67. In a physical sense, the clustering coefficient defines the extent to which nodes in the graph tend to form closely-knit groups that have many edges connecting each other in the group, but very few edges leading out of the group.

Definition 5 (Watts) The shortest path length $\bar{d}_v$ connecting each node $v \in V(N)$ of a network N to all other nodes is $d(v,j) \forall\, j \in V(N)$. The characteristic path length L is the median of all shortest paths.

The characteristic path length is a measure of the number of hops necessary to reach any node in the network from any other node. This indicates the degree of separation or connectivity between nodes in the graph. In Figure 4, node a can reach three of the nodes (b, d and e) through just one hop and the fourth node (c)

via two hops. So the characteristic path length for this node is:

$$\text{L}(a) = \frac{HopsToReachAllNodes}{NumberOfNodes} = \frac{(3\times1)+(1\times2)}{4} = 1.25$$

The characteristic path length (L) for the entire graph, which is the mean of the characteristic path length of all nodes, is equal to 1.2. The challenging as-

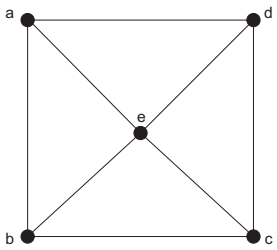


Fig. 4. Graph with $\gamma = 0.67$ and L = 1.2

pect in using small-world properties is that small-world networks combine the advantages of regular networks (high clustering coefficient) with the advantages of random networks (low characteristic path length).

C. Solution Encoding

Solution encoding is a major issue in this kind of algorithms since it will determine the choice of the genetic operators applied for exploring the search space. We have used a binary encoding of the solution in which each gene encodes an integer on 15 bits, that corresponds to one possible bypass link in the half-matrix of all possible links. For instance, if the maximum number of bypass links fixed a priori for the network that is optimized is 10, then for the genGA and the ssGA a chromosome will have 10 genes of 15 bits. Concerning CCGA, it will depend on the number of subpopulations used, if for instance CCGA is used with 5 subpopulations then one chromosome will have $\frac{10}{5} = 2$ genes of 15 bits. Figure 5 shows the example of a chromosome composed of 2 genes (thus the maximum number of created bypass links is 2) on a network of 5 stations. The 5×5 matrix represents all the possible links in the network including the already existing local links in the network (thus of the existing Wi-Fi connections) and the impossible links (i.e. links between two similar stations like station 1 - station 1) that are represented as shaded cells in the matrix (cells number 1, 7, 13, 19 and 25). In the example showed in Figure 5, the first gene (circled) with the integer value 3 stands for the connection between station 1 and station 3 in the corresponding matrix (also circled).

D. Fitness Function

As stated before we have used the Madhoc simulator to experiment injection networks optimization. Indeed, Madhoc allows to simulate and to visualize hybrid adhoc networks (using Wi-Fi, bluetooth, GSM, UMTS), to evaluate small world measures on them and to calculate the number of partitions in the network. In order

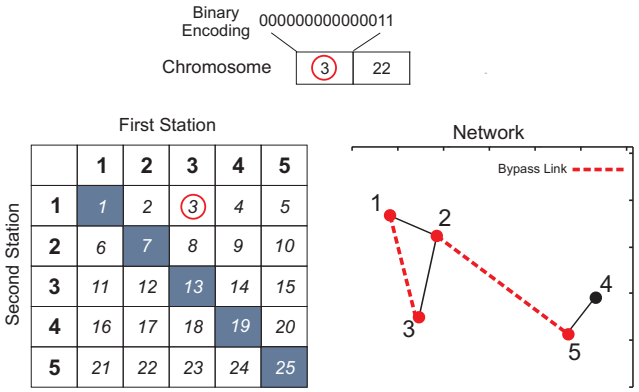


Fig. 5. Solution encoding example

to assign a fitness to the candidate solutions (i.e. sets of possible bypass links) of our algorithms, we use a unique cost function F which combines the two small world measures (L and γ) and the number of created bypass links. When computing the fitness function, we first test if the global network is connected. Indeed, since we use small-world properties as indicators, the network has to be connected in order to compute the characteristic path length (L) on the global network. Thus, if the optimized network is not connected, due to too few or not efficiently placed bypass links, the fitness value is a weighted term of the number of partitions in the network. On the contrary, if the network is connected, the fitness value is a linear combination of the small world measures (clustering coefficient and characteristic path length) and of the difference between the number of bypass links and the maximum number allowed. We look for maximizing the clustering coefficient and minimizing both characteristic path length and number of bypass links. Using this fitness function we thus have a maximization problem as defined in Algorithm IV-D.

Algorithm 1: Fitness Function

```
if Graph connected then
    F = α * γ + β * (L - 1) + δ * (bypassLinks -
    maxBypassLinks )
else
    fitness = 0.1 * numberOfPartitions
end
```

With weights experimentally defined:
$\alpha = 1$
$\beta = 1$ /(numberOfNodes -1)
$\delta = 2$ / (numberOfNodes * (numberOfNodes-1))

bypassLinks is the number of bypass links created in the simulated network by one solution, *maxBypassLinks* (defined a priori) is the maximum number of bypass links that can be created in the network, *numberOfPartitions* is the number of remaining partitions in the whole network after the addition of bypass links and *numberOfNodes* is the number of nodes in the global network.

V. Experimentations

This section presents the results obtained on the injection network optimization problem using the distributed CCGA compared to the results given by the generational GA (genGA) and the steady state GA (ssGA) . We first describe the parameters used for the three genetic algorithm. Next, the configuration of the network simulator is introduced and, finally the results obtained using the CCGA, genGA and ssGA are analyzed and compared.

The algorithms have been implemented in Java and tested on a single node for genGA, ssGA and CCGA and on 6 cluster-nodes for dCCGA (distributed CCGA) all nodes having a 3.7 GHz Xeon processor with 16 GB of RAM, running Debian Linux (with kernel 2.6.9-22) and Java version 1.5.0_05.

A. GA Parameterization

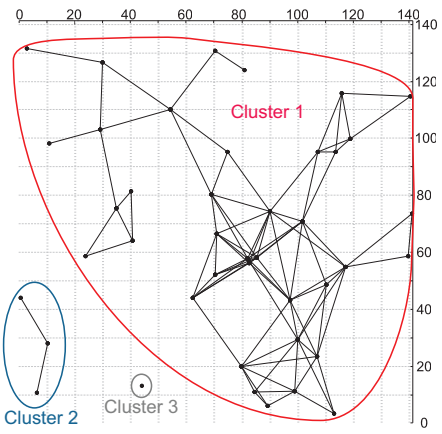


Fig. 6. Studied Networks with 3 clusters

In table I, we show the parameters used for genGA, ssGA, CCGA and dCCGA.

(d)CCGA was tested with 5 subpopulations. For all algorithms we used a randomly generated population composed of 100 individuals. The selection operator is a binary tournament selection (two individuals are selected and the fittest is copied into the intermediate population). The crossover operator is uniform crossover used with probability p_c=0.8. The mutation operator is bit flip mutation in which each allele of the chromosome is flipped with probability p_m= 1/chromosome_length. Concerning the generational GA and (d)CCGA we have added elitism: the best individual found in one generation is thus kept for the next generation.

TABLE I: Parameters used for genGA, ssGA, and (d)CCGA

Number of Subpopulations	5 (only for (d)CCGA)
(Sub)Population size	100 individuals
Termination Condition	50,000 function evaluations
Selection	Binary Tournament
Crossover operator	Uniform, p_c=0.8
Mutation operator	bit flip, p_m = 1/chrom_length
Elitism	1 individual (not for ssGA)

B. Madhoc Configuration

As stated before, the Madhoc simulator was used for managing the complex scenario posed by this injection network problem. We have defined a squared simulation area of 0.2 km^2 and tested with a density of 210 devices per squared kilometer. Each device is equipped with both Wi-Fi (802.11b) and UMTS technologies. The coverage radius of all mobile devices ranges between 20 and 40 meters in case of Wi-Fi. The studied network, as presented in Figure 6, here represents a snapshot of a mobile network in the moment in which a single set of users moved away from each other creating the clusters of terminals, that were obtained using the graphical mode of Madhoc. Used as example, the network with 3 clusters (center of Fig. 6) consists in 42 stations located in three partitions, the first partition has 38 nodes, the second one 3, and the third one has a single node. The number of possible connections in this 3-clusters network is $\frac{numberOfNodes^2}{2}$ = 882, the number of existing Wi-Fi connections in this network is 116, thus the number of possible bypass links is 882-116 = 766. The clusters are selected purposely to be different and thus challenging.

TABLE II: Parameterization used in Madhoc

Surface	0.2 km^2
Node Density	210 / km^2
Number of Nodes	42
Partitions	3
Possible Links	766

C. Results

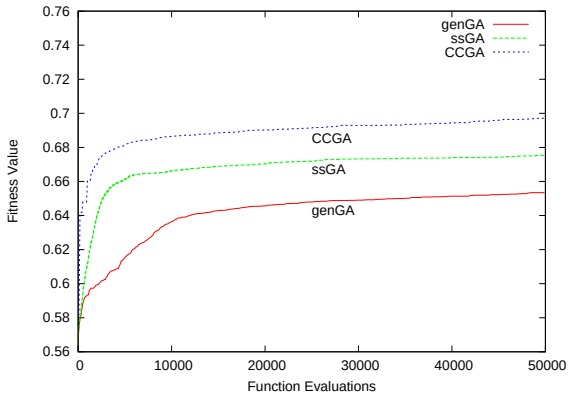


Fig. 7. Average results of 30 runs using genGA, ssGA and CCGA

Each result presented hereafter is the average obtained on 30 independent runs. In order to establish the statistical significance of the means, we first have checked that the data is normally distributed using the Kolmogorov-Smirnov test. If so, we then perform an ANOVA test so as to compare the means otherwise we use a Kruskal-Wallis test [20].

In Table III we show the averaged results for all 30 runs for each algorithm. As it can be seen in Table III, using a CCGA provides better results than both genGA and ssGA, genGA being the least performing

TABLE III: Results of all experiments

Network	GA	Crossover	Time	Result
3 Clusters	genGA	Uniform	58min 12s	0.6534
	ssGA	Uniform	70min 0s	0.6764
	CCGA	Uniform	138min 36s	**0.6971**
	dCCGA	Uniform	98min 55s	**0.6971**

one (with statistical confidence). This can be graphically observed in Figure 7, as well as the better convergence speed of CCGA compared to the other two GAs. As expected the computational time required for dCCGA is lower than for CCGA thanks to the distribution of the subpopulations, however it is still higher than panmictic GAs like genGA and ssGA due to the synchronization between subpopulation induced by the CCGA algorithm.

VI. Conclusion and Future Works

The results presented in this paper belong to an ongoing research on the injection network optimization problem using distributed coevolutionary genetic algorithms. Dafo, our distributed agent framework for evolutionary optimization, including coevolutionary genetic algorithms has been presented. The concept of injection network has been introduced as well as the utilization of small-world properties as indicators for inter-linking network partitions.

Experiments have been conducted using an ad-hoc network simulator on one network scenario composed of 42 stations. Three different GAs, generational, steady-state and cooperative coevolutionary, have been used, each one was tested using uniform crossover. The best result experimentally found on this problem, both in terms of best result found and convergence speed, was using the distributed CCGA. Initial evidence of the capacity of GAs and especially of coevolutionary GAs for solving this problem was also provided in this article.

As a future work, we plan to use some other coevolutionary GAs such as LCGA (Loosely Coupled Genetic Algorithm) to solve this problem. Our next research will also focus on the optimization of dynamic injection networks in which nodes move while optimum injection points are computed at the same time and thus bypass links have to be continuously created and destroyed in order to keep the network unpartitioned.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Technology and FEDER under contract TIN2005-08818-C04-01 (OPLINK project) and by the European CELTIC project (CARLINK).

References

[1] D. J. Watts, *Small Worlds – The Dynamics of Networks between Order and Randomness*. Princeton, New Jersey: Princeton University Press, 1999.

[2] T. Back, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Bristol, UK, UK: IOP Publishing Ltd., 1997.

[3] M. A. Potter and K. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature – PPSN III*. Berlin: Springer, 1994, pp. 249–257.

[4] R. Eriksson and B. Olsson, "Cooperative coevolution in inventory control optimisation," in *Proc. of the Third International Conference on Artificial Neural Networks and Genetic Algorithms*. University of East Anglia, Norwich, UK: Springer-Verlag, 1997.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[6] D. Whitley and J. Kauth, "GENITOR: A Different Genetic Algorithm," in *Proceedings of the Rocky Mountain Colorado, on Artificial Intelligence*, 1988, pp. 118–130.

[7] J. Paredis, "Coevolutionary life-time learning," in *Parallel Problem Solving from Nature – PPSN IV*. Berlin: Springer, 1996, pp. 72–80.

[8] F. Seredynski, A. Y. Zomaya, and P. Bouvry, "Function optimization with coevolutionary algorithms," in *Proc. of the International Intelligent Information Processing and Web Mining Conference*. Poland: Springer, 2003.

[9] O. Gutknecht and J. Ferber, "Madkit: a generic multi-agent platform," in *Proc. of the fourth international conference on Autonomous agents*. ACM Press, 2000, pp. 78–79.

[10] N. A. Barricelli, "Symbiogenetic evolution processes realized by artificial methods," *Methodos*, no. 9, pp. 35–36, 1957.

[11] P. Ratanchandani and R. Kravets, "A Hybrid Approach to Internet Connectivity for Mobile Ad Hoc Networks," in *Proceedings of IEEE WCNC*, 2003.

[12] A. Andronache, M. R. Brust, and S. Rothkugel, "Multimedia Content Distribution in Hybrid Wireless Networks Using Weighted Clustering," in *WMuNeP '06: Proceedings of the 2nd ACM international workshop on Wireless Multimedia Networking and Performance Modeling*. New York, NY, USA: ACM Press, 2006, pp. 1–10.

[13] N. I. T. Fujiwara and T. Watanabe, "A Hybrid Wireless Network Enhanced with Multihopping for Emergency Communications," in *ICC '04: Proceedings of the IEEE International Conference on Communications*, 2004, pp. 4177–4181.

[14] A. Helmy, "Small Large-Scale Wireless Networks: Mobility-Assisted Resource Discovery," 2002.

[15] O. Dousse, P. Thiran, and M. Hasler, "Connectivity in Ad-Hoc and Hybrid Networks," in *INFOCOM*, 2002.

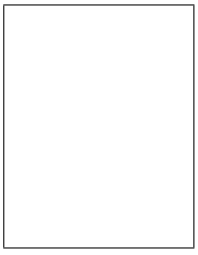
[16] D. Cavalcanti, D. Agrawal, J. Kelner, and D. F. H. Sadok, "Exploiting the Small-World Effect to Increase Connectivity in Wireless Ad Hoc Networks," in *ICT*, ser. Lecture Notes in Computer Science, J. N. de Souza, P. Dini, and P. Lorenz, Eds., vol. 3124. Springer, 2004, pp. 388–393.

[17] J. Li, C. Blake, D. S. D. Couto, H. I. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile Computing and Networking*. New York, NY, USA: ACM Press, 2001, pp. 61–69.

[18] K. Herrmann and K. Geihs, "Self-Organization in Mobile Ad hoc Networks based on the Dynamics of Interaction," Erlangen, Germany, 2003.

[19] L. Hogie, P. Bouvry, F. Guinand, G. Danoy, and E. Alba, "Simulating Realistic Mobility Models for Large Heterogeneous MANETS," in *Demo proceeding of the 9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'06)*. IEEE, October 2006.

[20] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.



Grégoire Danoy received the Industrial Engineer Degree in Computer Science from Luxembourg University of Applied Sciences in 2003 and the M.S. degree in Web Intelligence from Ecole des Mines of Saint-Etienne, France, in 2004. He is currently working toward the PhD degree with the University of Luxembourg and the Ecole des Mines of Saint-Etienne (France). His current research interests include nature inspired algorithms, multi-agent systems, dynamic optimization.

Bob++ : a Framework for Exact Combinatorial Optimization Methods on Parallel Machines

François Galea and Bertrand Le Cun
PR*i*SM Laboratory, University of Versailles
45 avenue des Etats-Unis, 78035, Versailles, France
Francois.Galea@prism.uvsq.fr, Bertrand.Lecun@prism.uvsq.fr

Abstract— **The aim of this article is to propose the object-oriented design of the Bob++ framework. Bob++ is a framework for implementing solvers for combinatorial optimization problems on parallel and sequential machines. Several similar frameworks have been proposed in the last decade but each of them only focuses in one method, said Branch-and-Bound, Divide-and-Conquer, etc.. and proposes also one parallelization, which is very difficult to extend. We propose a software design where: first, several exact combinatorial optimization methods are made available to the user to solve a problem, and second, an interface to facilitate the implementation of a parallelization is also provided. Parallelizations may use POSIX threads as well as MPI, or more specialized libraries such as Athapascan/Kaapi.**

Keywords— **Combinatorial Optimization, Search algorithms, Branch-and-Bound, Parallelism, Cluster, Grid Computing**

I. Introduction

Large scale decision and optimization problems belong to the class of the best applications for parallel machines and also for computational grids. The problems are in the NP-Hard complexity class and may require an exponential computational time in the worst case. It is natural to consider the parallelization of the search process when a solution of a large-scale problem is out of reach when using a single-processor computer.

The tremendous attention that the parallelization of these methods as Branch-and-Bound has received in the literature gives some indication of its importance in many research areas.

But as in many domains, several software frameworks have been proposed, establishing the interface between the users and the parallel machine. These tools include Bob++ [21], BCP [13], PICO [3], ALPS [20], [12], [18], Bob [9], PUBB [15], [14], PPBB [19] ....

It is possible to classify these different existing frameworks according to two major criteria:

1. The **node search algorithm** involved in the search process. These algorithms include Branch-and-Bound (B&B), Divide-and-Conquer (D&C), A* and Dynamic Programming (DP).
2. The **programming environment** they use to implement the parallelization. Some of the available programming environments are POSIX threads, MPI, PVM, and Athapascan/Kaapi.

Many of the available parallel search algorithm frameworks are specialized at the same time in the algorithm they implement, and for a specific programming environment. For example, BCP [13] is an implementation of the Branch-and-Price-and-Cut algorithm, which runs on the MPI programming environment. PICO [3] is a Mixed-integer solver, which implements B&B, and also runs on MPI.

Some other projects tend to diversify some aspects of the solver framework. SYMPHONY [17], for example, solves mixed-integer programming (MIP) problems using PVM for distributed memory machines or OpenMP for shared memory machines. ALPS [20], [12], [18], which in some way is a successor of SYMPHONY [17] and BCP [13], generalizes the node search to any tree search, which of course enables B&B search, among others. Though, the only available programming environment for ALPS [20], [18] is MPI. In a similar manner, PEBBL [4] integrates the B&B search from PICO [3], allowing the implementation of a larger variety of solvers than MIP solvers.

The old version Bob [9], [11] focus on parallel Branch and Bound. Others methods like A* have been added but with awful hacks. What Bob++ proposes is to provide different search algorithm classes, while being able to use different possible parallelization methods. The goal is to propose a single framework for most classes of combinatorial optimization problems, which can be solved on as many different programming environments as possible. Figure 1 shows how Bob++ interfaces between high-level applications (QAP, TSP, ...) and different possible parallel programming environments. However, Bob++ is still under developement.

Bob++ has been developed in C++ language, and proposes a C++ API, composed of basic classes which are extendable by the user.

Most real-life tests have been done using the Branch-and-Bound search algorithm, showing the robustness of the Branch-and-Bound application interface. Most of other developed applications are just validation tests for the design of some of the node search algorithms, such as a simple N-Queens problem solver which uses Divide-and-Conquer.

Most of recent work has been focused on Branch-and-Bound. Dynamic Programming and A* are currently unavailable, due to changes we made in the Bob++ structure when developing Branch-and-Bound and Divide-and-Conquer, even though these changes have been done with the addition of Dynamic Programming and A* in mind. This is why the only application interface we will talk about in the following sections of this article is Branch-and-Bound.

The next section deals with the Bob++ application interface. The parallel interface is presented in the sec-

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

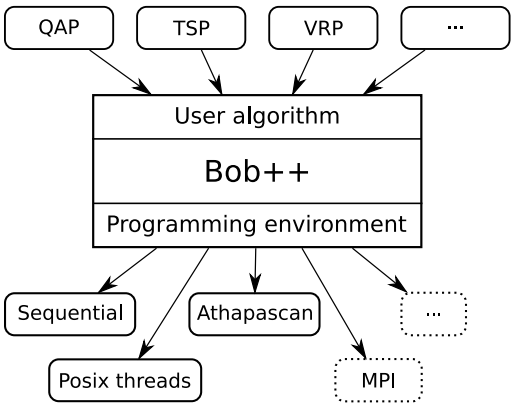


Fig. 1. The structure of Bob++ applications

tion III. The section IV shows two applications. Concluding remarks and future work are presented in the section V.

II. Application Interface

The idea behind the Bob++ application interface is to provide the programmer with an easy interface for programming parallel node search algorithms, based on the following classical methods : A*, Divide-and-Conquer, Dynamic programming and Branch-and-bound. At this time, the only implemented methods are Divide-and-Conquer and Branch-and-Bound.

In Bob++, each of the mentioned methods is implemented by four classes. The base classes for each of them are the following:

- The `Instance` class is used for the storage of all the global data of the application. This data is initialized at the beginning of the resolution. It is not allowed to modify its contents during the execution of the resolution.
- The `Node` class enables the storage of the data and contains the methods associated to a node in the search space.
- The `GenChild` class contains the method used to generate the child nodes of a given node.
- The `Algo` class contains the execution code of the main loop of the resolution.

Each of these base classes are derived into specific classes which are specialized for the resolution of the different possible search algorithms.

The specialized `Instance`, `Node` and `GenChild` classes are called "the user classes", meaning that these classes must be customized by the user to implement the resolution of its own problem. The virtual methods that the user needs to redefined, are different according to the choosen method (B&B, D&C, ...).

In order to perform strong type checking at compilation time and to avoid downcast, Bob++ makes an exhaustive use of templates in the class definition. The Bob++ classes are parametrized by a `Trait` class which only contains types definitions. The `Trait` must contain the definition for the `Node`, the `Instance`, the `GenChild` and the `Algo` but also for the `Stat` class and `Priority` class. This modelization, is widely used in C++ Standard Template Library (STL). The figure 2 shows a short example of this.

```
class MyTrait {
  public:
    typedef MyNode Node;
    typedef MyInstance Instance;
    typedef MyGenChild GenChild;
    typedef Bob::BBAlgo<MyTrait> Algo;
    typedef Bob::BestEPri<MyNode> PriComp;
    typedef Bob::BBStat Stat;
};

class MyNode : public Bob::BBIntMinNode {
    ....
};

class MyInstance : public Bob::BBInstance<MyTrait> {
    ....
};

class MyGenChild: public Bob::BBGenChild<MyTrait> {
    ....
};
```

Fig. 2. Example of a `Trait` class

A `Stat` instance stores all the activities of an associated `Algo`. It is used for monitoring the execution of the resolution, either in a offline or online way. The user can extend the definition of the default `Stat` class to add statistics or monitored values which correspond to its specific needs.

The `Priority` class contains the rules to schedule the search. Bob++ provides default classes derived from `Priority`. These default classes can be used when a standard node selection rule (depth first, best evaluation first, etc.) is enough. The user can also choose to create a specific `Priority`-derived class.

A. The log system

Using the `Stat` class, or any of its derived classes, makes possible to the user to get a lot of information from the execution of the algorithm.

The `Stat` is instanciated only once in sequential and shared memory environments, hence the generated log data is global in this case. In distributed memory environments, one `Stat` instance is associated to each running `Algo` instance, which generally corresponds to a processor of the parallel machine. Thus, the different `Stat` objects only store locally-generated data.

While the algorithm is running, a `Stat` object generates statistics about general data such as the number of generated nodes, evaluated nodes, or the number of calls to the generation method of the `GenChild`, as well as problem-specific information. It is possible to redirect the output of a `Stat` object to a file, hence the execution evolution can be analyzed after the execution is finished, or during the execution.

It is also possible to view the data from the file when the execution is not finished. Of course, in this case, all the processors must output their logging information to the same file, thus this behaviour is limited to the sequential and multi-threaded programming environments.

In shared memory programming environments, the

analysis of the contents of a log file allows to obtain both global and processor-specific information about the execution of the algorithm. In distributed memory programming environments, only processor-specific data is available in the different log files. This why the log system offers the possiblilty to configure a `Stat` instance to output its data to a network host machine. This host simply runs a *boblistener* program, which listens for incoming log information from the different parallel computer nodes, and gathers them to a single log file. This way, it remains possible to have real-time access to centralized global and processor-specific information, even in distributed memory environments.

In order to display the generated log information, a tool called *bobview* has been developed. This tool can display the contents of a log file, either after the execution of the algorithm, or while it is running. As the log information contains both global and processor-specific data, *bobview* offers different view modes for the analysis of the execution.

As an example, figure 3 shows the displayed information which is generated by the execution of the MIP solver running on a dual processor shared memory computer, using the multi-threaded programming environment. The first view displays global information about the global priority queue activities, basically the evolution of the number of priority queue insertions, deletions, and number of stored nodes. The two other plots display thread-specific information.

This log system has been developed with the idea that the user should be able to easily get information feedback from the behaviour of its algorithms. This information allows easy tuning of an application without having to wait for the end of its execution.

B. The B&B abstract solver

A solver using the B&B method is written by extending the 3 following "user" classes: `BBNode`, `BBInstance` and `BBGenchild`.

To be able to compare two `BBNode` objects according to their evaluation, the `BBNode` class is parametrized with the sense of optimization (maximisation or minimsation) and the type of the evaluation (int, float, double, etc..). Shortcuts have been predefined, such that the user can choose one the different available default node classes i.e. `BBMinIntNode`, `BBFloatMaxNode` ... as a base class for its own `Node` class. In Bob++, only one type, one class is used to represent the subproblem or a solution of the problem.

The `BBInstance` class must be extended to enable the storage of all the information needed for the resolution of the specific problem. For example when solving a MIP problem, one may plan to include the original MIP problem in the `BBInstance`-derived object, in order to have constant access to the knowledge about integer and continuous variables.

The `BBGenChild` class is not really different from the basic `GenChild` class. The user must extend the `BBGenChild` class to redefine the `operator()` which performs the branching operation from a `Node` object.

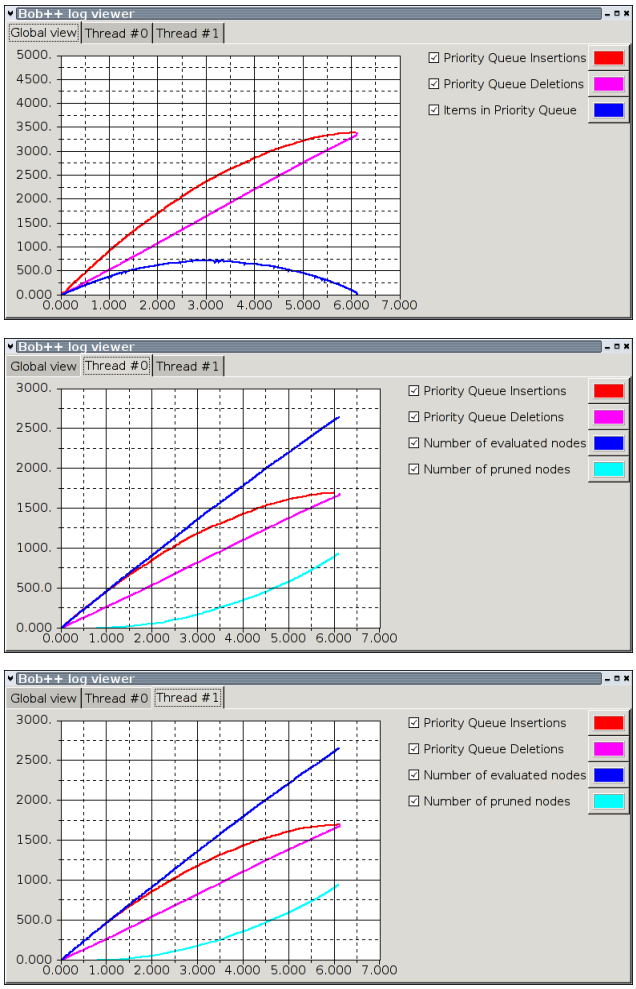


Fig. 3. Execution Analysis of a multi-threaded B&B application

The user code must ensure that each newly-generated child `Node` object which is suitable to be explored is inserted into the priority queue if it is a subproblem.

A `Priority` class is associated to the specialized `Node` class. The `Priority` class must be defined in the `Trait` class.

The figure 4 shows the UML sequence diagram of the search procedure i.e. the `BBAlgo::operator()`.

III. Parallel Interface

Unlike the other frameworks, the Bob++ design integrates a programming interface to implement all kinds of tree search parallelizations. Many of the existing frameworks propose only one parallelization. For example, PICO [3] and ALPS [20], [12], [18] propose a Master-Hub-Worker parallelization, SYMPHONY [17] uses a Master/Worker paradigm using PVM.

There exists a large variety of machines and a large variety of problems. One could be able to use a dual-core processor, as well as cluster of workstations. Bob++ could be extended using this interface by adding a new *Programming environment*. At this time, the proposed environments are sequential, multi-threaded (making use of POSIX threads) and Kaapi-based environments. An MPI version is currently under development.

A. The Programming Environment principles

The main idea leading the Bob++ design to obtain parallel solvers is a generalization of the *Global priority Queue (GPQ)* used to implement the old Bob [9], [11] library. The principle is to have different instances of `Algo` that are executed on different processors. These different `Algo` communicate, and are synchronized using high level communication tools. These tools are mainly the data structures that store the subproblems, the solutions or other information needed by a specific method. These *Global Data Structures* are equivalent to the Knowledge Pool introduced by ALPS [20], [12], [18].

For example in the context of a B&B, these "parallel" instances of `BBAlgo` will mainly execute a loop where at each iteration the `operator()` of an instance of the user `BBGenChild` is executed on a pending node obtained from a *Global priority Queue* `GPQ`. The new generated nodes are re-inserted in `GPQ`. In the same way if a solution is found, the data structure called *Global Solution* `GSol` will be updated with this new solution.

The different `GPQ` and `GSol` must communicate to ensure that each `Algo` has enough work to do and has the latest updated Solution.

The `GPQ` and `GSol` can be considered from the `Algo` point of view as high level communication tools since the different `Algo` instances only communicate through these tools.

This design does not constraint the algorithm used to manage the nodes or broadcast a new solution value. It does not imply a specific parallel strategy. The goal is to be able to implement, a Master/Hub/Worker strategy used by PICO [3], PEBBL [4] and ALPS [20], [12], [18], but also simple Master/Slave strategy using MPI, or different parallelization strategies using the POSIX threads on a shared memory machine. One could also implement a parallelization where the load balancing strategy takes into account the heterogeneity of a Grid. The figure 4 shows the UML sequence diagram of the `BBAlgo::operator()`, which performs the main loop of the algorithm. The `GPQ` and the `GSol` are respectively called `ThrPQ` and `ThrSol` which are concrete classes of the mutlithreaded environment.

An interesting property of this design is also that the parallel and sequential `Algo` implementations are exactly the same. The only difference is the different implementation of the *Global Data Structures*.

Abstract classes are proposed in the Bob++ Framework, to define the interfaces needed by the `Algo`. The `PQInterface` defines the interface used by the B&B algorithm to store the pending nodes. Concrete classes are also defined. For example the `PQSkew` class extends the `PQInterface` interface. In this case, the algorithm used to store the nodes according to their priorities is the Skew-Heap [16].

The *Threaded* programming environment defines another concrete class which extends the `PQInterface` called the `ThrPQ` (see figure 4). The goal of this class is to enable the access to the `PQSkew` in mutual exclusion mode. The figure 5 shows the UML sequence diagram

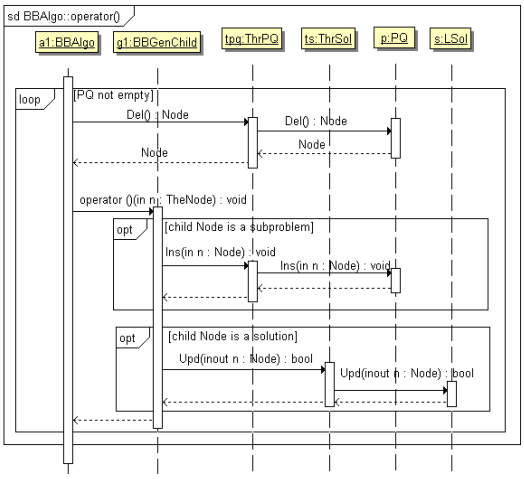


Fig. 4. The UML sequence diagram of a search

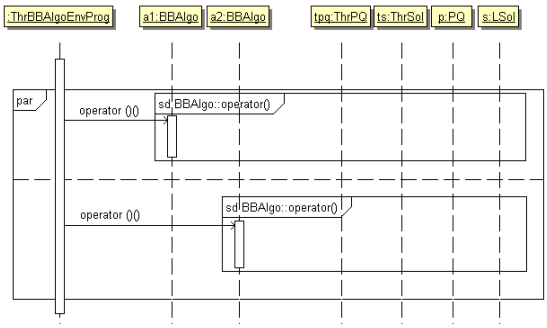


Fig. 5. The UML sequence diagram of multithreaded search

of the threaded environment, which calls in parallel the Search sequence diagram (see figure 4 on 2 differents instances of the `BBAlgo` class. In this diagram, the two instances of the BBAlgo use the same instance of `ThrPQ` and the same instance of `ThrSol`.

Therefore, the initialisation of an `Algo` e.g. the initialisation of an instance of the `Algo` itself and the initialization of the data structure it uses, are performed by the *Programming Environment*. The Bob++ Programming Environment class which realizes all this initialization is the `AlgoEnvProg` class.

More general initialization than which is made on the `AlgoEnvProg` is often necessary for a specific environment. For example, in a context of a multithreaded environment (called *Thr*), the different threads must be created, but after the parsing of the command line. These initializations are usually made in *static* methods. The choice between the environments is made in the *main* function. Fig. 6 shows an example of a *main* function that can be used to generate different solvers for the same problem, making use of different programming environments.

The multithreaded environment has been proposed in order to propose an easy and powerful use of modern dual- or quad-core processors. It also constitutes the first step for the implementation of a hybrid algorithm for clusters of shared-memory machines.

```
int main(int argc, char **argv) {
#ifdef Atha
   Bob::AthaBBAlgoEnvProg<MyTrait> env;
   Bob::AthaEnvProg::init(n,v);
   Bob::core::Config(n,v);
#elif defined(Threaded)
   Bob::ThrBBAlgoEnvProg<MyTrait> env;
   Bob::ThrEnvProg::init();
   Bob::core::Config(n,v);
   Bob::ThrEnvProg::start();
#else
   Bob::SeqBBAlgoEnvProg<MyTrait> env;
   Bob::core::Config(n,v);
#endif
   MyInstance *Instance=new MyInstance( );
   env(Instance);
#ifdef Atha
   Bob::AthaEnvProg::end();
#elif defined(Threaded)
   Bob::ThrEnvProg::stop();
#endif
   Bob::core::End();
   delete Instance;
}
```

Fig. 6. Example of a `main` function

B. The Athapascan Environment

We use the Athapascan/Kaapi library [23] as a parallel environment for Bob++. Athapascan/Kaapi is a high level parallel programming tool. Its aim is to schedule a set of tasks which are created dynamically using Athapascan's *Fork* primitive. Athapascan/Kaapi has been developed in such a way that one does not have to worry about the specific machine architecture or the optimization of load balancing between processors. The created tasks are scheduled on the different processors in order to complete the work.

In the context of B&B, the Athapascan/Kaapi library is a very interesting tool to parallelize the search procedure. A strategy consists in creating a task to explore a subtree rooted on a node. The leaves of the subtree which are subproblems become new tasks. The idea is to have enough tasks to ensure that all processors have enough work. In Bob++, the task that explores a subtree is a full instance of a Bob++ `Algo`. The Athapascan primitive used to `fork` a task is called in the `GPQ`'s *insert* method. Different strategies are defined in order to have different sizes of subtrees according to the position of the root node in the tree. When the node is very close to the root node of the B&B tree, the size of the subtree explored by a task must be very small, to produce work very quickly. When the root node of a subtree is on the middle of the B&B tree, the size of the subtree could increase. The Athapascan/Kaapi runtime stores the list of waiting tasks in a list which is local to each process. If the process runs on a machine that has several processors, the list is shared among the threads that are running on each processor and that execute the tasks. The next section shows some results with this environment.

IV. Bob++ Applications

This section describes the more advanced applications we designed on top of Bob++.

Instance	# Procs	Time (mn)
Nugent17	168	0.33
Nugent20	50	5.69
Nugent20	168	3.58
Nugent20	184	3.32
Nugent21	50	11.5
Nugent21	168	7.19
Nugent22	50	9.8
Nugent24	140	153.7

TABLE I: Tests of the QAP code

A. The QAP solver

The first Branch-and-Bound application is a solver for the Quadratic Assignment Problem. The QAP was formulated by Koopmans and Beckmann (1957) [8] as the task to find a permutation π of $\{1, ..., n\}$ that minimizes:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} C_{ijkl}x_{ij}x_{kl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \quad (1)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n \quad (2)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., n \quad (3)$$

$$x_{ij} = 0, 1 \quad i, j = 1, .., n \quad (4)$$

where C_{ijkl} denotes the cost incurred by locating facility i on location j and facility k on location l. It represents the product of f_{ik} (flow between the facilities i and k) and d_{jl} (distance between the locations j and l). The QAP Solver on top of Bob++ is based upon the dual procedure of Hahn and Grant [6] to compute the lower bound and the polytomic branching strategy of Mautor and Roucairol [10]. This code has been tested on various machines [2], [1] using the Athapascan/Kaapi parallelization. In [7] this code with Bob++ has been tested with a fault tolerant version of Athapascan/Kaapi. Table I shows the results of the QAP code on the some Nugent instances from the QAPLIB. The runs have been realized on a Itanium-2 cluster of the university of Grenoble. The CPU times show that the algorithm is scalable.

B. The MIP solver

The second Branch-and-Bound application which has been developed is a simple Mixed-Integer Programming (MIP) solver. The solver can take the description of a problem from a *CPLEX* LP file, then solve it using the chosen parallel environment. The LP solver used during the node evaluation can be chosen from different existing solvers : *CPLEX* from Ilog or *Xpress-MP* by Dash Optimization are the possible available commercial solvers, but solvers from the open source community can also be used : *GLPK*, *LP_solve* and *Clp*.

For the moment, the integrated MIP solver is at a very early stage of development. It uses simple

branching methods, and does not make use of generic cuts. Therefore, it cannot be compared to the available efficiency-proven MIP solvers. However, we obtain very good speed-up improvements using the POSIX threads environment, on multi-processor shared-memory architectures. Our objective for the near future is to perform heavy testings on distributed memory environments, as soon as the necessary code is added to the MIP solver.

The flexibility for the LP solver choice has been made possible by using the *Glop* library which we developed ([5], [22]). *Glop* is a free-software application programming interface (API), which wraps solver-specific LP or MIP solver function calls. Its base idea is to provide the user with a generalized API which allows to suppress the dependence of the code to a specific solver. This way, it is for example possible to compare the performance of the different solvers by using the same optimization code.

Another solver interface from the open-source community already exists : *Osi* from the COIN OR project [13]. *Glop* differs from *Osi* to the fact that it is only a lightweight wrapper to the solver API calls, whereas *Osi* is a C++ interface which allows the problem modelization and resolution in an object-oriented way, which is not the usual way solver APIs usually work.

V. Summary and Future Work

In this paper, we have presented the main features of the Bob++ Framework. We described the User Programming Interface and the Parallel Programming Interface, called the Programming Environment interface. We explain how is it possible to use a high level parallel library in Bob++. We show the good performance we obtained on the QAP problem using the Athapascan/Kaapi library. We also quickly present the MIP solver that uses the *Glop* library using the Multithreaded environment. Its development is still in progress, this is why no good results are currently available. The current solver can not be compared to other existing solvers as it still lacks cut generation.

We present the log system of Bob++, which provides the ability to analyze the execution of the parallel algorithm. The statistics are available during program execution, even in distributed memory environments. They can also be analyzed after the execution.

Current work includes the addition of generic cut generation to the MIP solver, which as we expect should provide significant performance increase.

Several other features could be proposed for the Application Interface and to increase the Parallel Interface. As said in the introduction, we would like Bob++ to propose interfaces for other methods as Dynamic Programming and A*. Higher level interfaces could be proposed to facilitate the programming of more specialized algorithms built on top of B&B, like Branch-and-Price and Branch-and-Cut. An old version of Bob++ used to include these features: we have to port them on this current version. The parallel interface could be extended to propose other parallel library ports like MPI.

References

[1] A. Djerrah. *Résolution Exacte d'un problème d'optimisation combinatoire NP-difficile sur grilles de machines*. PhD thesis, Université de Versailles-Saint-Quentin, july 2006. In French.

[2] A. Djerrah, S. Jafar, V.-D. Cung, and P. Hahn. Solving QAP on cluster with a bound of reformulation linearization techniques. In *In the 17th IMACS World Congress Scientific Computation, Applied Mathematics and Simulation IMACS2005*, Paris, France, July 2005.

[3] J. Eckstein, C. A. Phillips, and W. E. Hart. PICO: An object-oriented framework for parallel branch and bound. In E. Scientific, editor, *Proceedings of the Workshop on Inherently Parallel Algorithms in Optimization and Feasibility and their Applications*, Studies in Computational Mathematics, pages 219–265, 2001.

[4] J. Eckstein, C. A. Phillips, and W. E. Hart. PEBBL 1.0 User Guide. RRR 19-2006, RUTCOR, August 2006.

[5] F. Galea. *Problèmes d'optimisation en curiethérapie*. PhD thesis, Université de Versailles–Saint-Quentin-en-Yvelines – UVSQ, 45, Avenue des Etats-Unis, 78035 Versailles Cedex, FRANCE, Sept. 2006. In French.

[6] P. Hahn and T. Grant. Lower bounds for the quadratic assignment problem based upon a dual formulation. *Operations Research*, 46:912–922, 1998.

[7] S. Jafar. *Programmation des systèmes parallèles distribués : tolérance aux pannes, résilience et adaptabilité*. PhD thesis, INP de Grenoble, June 2006. In French.

[8] T. Koopmans and M. Beckman. Assignment Problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[9] B. Le Cun, C. Roucairol, and the PNN team. Bob : a unified platform for implementing branch-and-bound like algorithms. RR 95/16, Laboratoire PR*i*SM, Université de Versailles - Saint Quentin en Yvelines, Sept. 1995.

[10] T. Mautor and C. Roucairol. Difficulties of Exact Methods for Solving the Quadratic Assignment Problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *Discrete Mathematics and Theoretical Computer Science*, pages 263–274. DIMACS, American Mathematical Society, May 1994.

[11] M.Benaichouche, V.Cung, S.Dovaji, B. Cun, T.Mautor, and C.Roucairol. *Building a Parallel Branch and Bound Library*, volume 1024 of *LNCS State-of-the-Art Survey*, pages 201–231. Springer-Verlag, 1996.

[12] T. Ralphs, L. Ladnyi, and M. Saltzman. A Library Hierarchy for Implementing Scalable Parallel Search algorithms. *The Journal of Supercomputing*, 28(2):215–234, may 2004.

[13] M. J. Saltzman. *COIN-OR: An Open Source Library for optimization*. Kluwer, Boston, 2002.

[14] Y. Shinano, M. Higaki, and R. Hirabayashi. An Interface Design for General Parallel Branch-and-Bound Algorithms. In *Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 277–284, 1996.

[15] Y. Shinano, M. Higari, and R. Hirabayashi. Generalized utility for parallel branch-and-bound algorithms. In *Proceedings of the 1995 Seventh Symposium on Parallel and Distributed Processing*, number 392, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[16] D. Sleator and R. Tarjan. Self-Adjusting Heaps. *SIAM J. Comput.*, 15(1):52–69, Feb. 1986.

[17] T.K.Ralphs and M. Guzelsoy. The SYMPHONY Callable Library for Mixed Integer Programming. In *In proceedings of the Ninth INFORMS Computing Society Conference*, 2005.

[18] T.K.Ralphs, L. Ladányi, and M. Saltzman. Parallel Branch, Cut, and Price for Large-scale Discrete Optimization. *Mathematical Programming*, 98(253), 2003.

[19] S. Tschoke and T. Polzer. Portable parallel branch-and-bound library user manual, library version 2.0. Technical report, Department of Computer Sciences, University of Paderborn., 1996.

[20] Y. Xu, T.K.Ralphs, L. Ladnyi, and M. Saltzman. ALPS: A Framework for Implementing Parallel Search Algorithms. In *In proceedings of the Ninth INFORMS Computing Society Conference*, 2005.

[21] Bob++: Framework to solve Combinatorial Optimization

Problems
http://bobpp.prism.uvsq.fr/.
[22] Glop, the Generic Linear Optimization Package.
http://glop.prism.uvsq.fr/.
[23] Kaapi: a library for high performance parallel computing based on an abstract representation to adapt the computation to the available computing ressources.
http://kaapi.gforge.inria.fr/.

A Grid-enabled Framework for Exact Optimization Algorithms

I. Zunino, N. Melab and E-G. Talbi
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59655 - Villeneuve d'Ascq Cedex - France
E-mail: izunino@exa.unicen.edu.ar; {melab,talbi}@lifl.fr

Abstract

In this paper we present a framework for writing exact optimization algorithms distributed on a grid environment. It presents a new way of reusing design and code for multi-objective optimization methods in conjunction with assistant methods. These kinds of methods are used mainly for reducing the search space, or for using a mono-objective method for solving a multi-objective problem, or both. We use a master-slave paradigm for the parallelization of the work units and a branch and bound algorithm as a default assistant method. The branch and bound algorithm is also distributed on grids which allows a two level parallelism for the optimization. We show how the different objects are codified in order to allow less communication while at the same time maintaining the reusability requirement. A sample instantiation of the framework is presented using the Parallel Partitioning Method (PPM). Preliminary results are shown using different Flowshop instances.

Keywords: *Frameworks, Branch and Bound, Parallel Computing, Grid Computing, Multi-Objective Optimization, Flow-Shop Problem*

1 Introduction

There exist a grand variety of frameworks for exact optimization. Most of these frameworks use a branch and bound algorithm to perform the search, or facilitate the programmer in writing branch and bound-based algorithms. Examples of such frameworks are: PUBB [19], BOB++ [2], PPBB [16], PICO [7], MALLBA [4], ZRAM [1], ALPS [20], MW Framework [8], Symphony [14]. There is a good taxonomy of parallel software frameworks and an overview of the implementation of parallel branch and bound algorithms and frameworks in [15].

In multi-objective optimization there exist different methods or strategies to optimize without searching in all the search space. Examples of methods which use those strategies are the Two Phases Method (TPM)[17], the Parallel Partitioning Method (PPM)[9] and the epsilon-constrained method[18]. These methods have the particularity of optimizing using a strategy. This strategy is used, in order to reduce the search space, for using a mono-objective method for solving the multi-objective problem, or both. In order to do this, it uses another method (an assistant method) to optimize different subspaces. Hence, each time one of these methods is written, a new implementation of the assistant method has to be written or at least, the connection between the strategy and the method has to be built. At the moment, to the best of our knowledge, there are no distributed frameworks which address directly the use of methods for multi-objective optimization by the assistance of another method. Our framework addresses these kinds of problems using a default assistant method and at the same time being distributed on a grid environment. In our work, we will be having a default implementation using as an assistant method the branch and bound algorithm developed in [12]. This algorithm can be used with different problems, and accepts multi-objective and mono-objective optimization. It has proven to be very efficient and highly scalable on a grid environment [12].

The rest of the paper is organized as follows: Section 2 presents the design requirements and objectives and the overall architecture of the proposed framework from design and implementation points of view. Section 3 describes the application of the framework for solving the bi-objective Flow-Shop scheduling problem and its experimentation on the Grid5000 French experimental grid. In Section 4, the conclusion and perspectives of the work are drawn.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

2 The Framework

Usually, a family of related applications has many of their functionality similar, if not the same. A framework addresses this aspect by providing abstract representations of classes and implements invariant parts for the different applications in a family. More specifically, "A framework is a set of abstract classes and components that together comprises an abstract design solution for a family of related applications" [10].

2.1 Design requirements and objectives

A framework lets us make different applications within a domain of functionality. In order to be able to do that, it is needed that we specifically define the domain. The functionality the framework is expected to satisfy is called functional requirements. There are also quality criteria which the framework should try to satisfy and they are called non-functional requirements.

Functional requirements

- The framework should facilitate the implementation of different exact optimization methods for exploring the search space.
- There should provide parallel support for the different parallel exact optimization methods that need it such as TPM or PPM.
- It should be able to distribute the provided parallelism on grids.

Non-functional requirements

- Reusability: the code and design solutions should be reusable.
- Modifiability: the development of new methods should be done in a clear way concerning the underlying optimization method assistant and parallel behavior. That is, the programmer should have to write only the part of the new method.
- Flexibility: the framework should be easily extended to provide the programmer with the ability to easily instantiate different optimization domains. That is, using different objectives, restrictions or models for the optimization.
- Extensibility: It should be easily extended in order to support new parallel or distributed technologies without the need of changing the code for the optimization algorithms or search strategies.
- Performance: it should provide by default with high-performance optimization algorithms so that the users can focus only on the details concerning their new search strategy for optimization. The distributed and parallel requirements should be met without degrading the performance of the system.
- Scalability: As the framework should be distributed on a grid environment where a large number of processors may be available, it is important that the performance is improved as the number of processors used grows.

2.2 System Architecture

In order to satisfy the requirements above, we implement different design decisions going from an architecture point of view to a design point of view. The architecture lets us address most of the functional and non functional requirements by providing a general view of the interactions of the different modules inside the framework without the need of a detailed specification. The design lets us see more in detail how the framework will be used and instantiated.

Concerning the goals for modifiability and extensibility, we think that working with a layers style makes a strong separation of functionality so that the change in one of the main features would conclude in changing as less code as possible or nothing at all for the other layers. Concerning the functional requirement for parallel search algorithms, we provide with a layer that implements the functions for optimization using asynchronous communication so that each call to those functions doesn't stop the flow of the main structure of the optimization method. The goal of distributing the work on a grid environment is addressed using components with distributed capabilities, that is, they should be serialized (codified) and deserialized (decodified) in order to be sent through the grid. At the same time, with the purpose of making the framework as less attached as possible to any technology, this is implemented in a different layer. The framework provides with a default implementation of this layer and we think making it work with the Message Passign Interface (MPI) is a good option since the programs can be deployed on a cluster or on a grid environment.

The first design decision we take is the separation of the main method for exploring in the search space (i.e.: TPM, epsilon-constrained, PPM) from the solver that performs the optimization within a particular space (i.e. B&B). The StrategySolver layer has the responsibility of optimizing the space in which to explore for the solutions. In order to find these solutions, it uses the OptimizationSolver layer which responsibility is the optimization of an objective function inside a given space. In this way, each time a new

algorithm is written, there is no need to make any changes to the optimization code.

As it was previously said, the StrategySolver should be able to work in parallel for some of its functions. This is achieved by making the OptimizationSolver able to receive and work with asynchronous calls. So, each time the StrategySolver has to perform different optimization tasks in parallel, it calls the OptimizationSolver as many times as needed.

Concerning the distribution of the paralellism there is another layer that performs the distributed calls so that the optimization layer may take place in different machines. By default, this is done using the MPI layer. The architectural view is represented in Figure 1. In this view all the layers of the MPI API are not shown and depend on the implementation. However, as it was previously said in the extensibility requirement, this layer can be changed without the need of changing the implementation of either the strategy or the optimization algorithm used, as long as it conforms to the same interfaces.

A Sequence Diagram used to show the actions used during the distribution of optimization requests to the different slaves is shown in Figure 2. In this example, we can see the strategy (StrategySolver) calls asynchronously (the flow of the method is not interrupted) the master (MasterMPISolver) two times to optimize . At each time, the MasterMPISolver calls one different Slave and after that it remains waiting for the answer in the method getSolutions(). Then, each of the slaves call its own solver which would actually perform the optimization. After the internal solvers finish optimizing, each slave sends the results to the Master. When all the results arrive the master returns the answers to the strategy method. In this Figure, in order to make it readable, we hide the part of the codification of the objects that where distributed. The codification will be shown later.

2.3 Grid-based implementation

A summary of the main classes used for the optimization process is represented in Figure 3. The class OptimizationSolver is the abstract class used to represent the main method and the assistant method (i.e. the branch and bound). This solver has a method to perform optimization with Objectives, Restrictions and InitialSolutions. Each of these components is represented with a different class. The Objective class has a method which returns the evaluation of that solution for a specified objective. The Restriction class has also an evaluation method and it also works with solutions, but returns a boolean value indicating whether the solution is inside the feasible space or not. The initialSolutions can be used to feed the solver or the strategy with initial solutions for optimizing the search. This is also used in some of the strategies as we shall see later in the example of PPM.

For the distribution part, we implement a wrapper for the OptimizationSolver which acts as a proxy, redirecting the different asynchronous calls to different machines. This wrapper is represented in both classes: MasterMPISolver and SlaveMPISolver. They use a master-worker paradigm to distribute the work. The Master is connected to the main method or strategy and the Solver is connected to the assistant method. In our default implementation using the branch and bound proposed in [12], the assistant method is also distributed on the grid so this generates a two level paralellism and distribution which we believe can be helpful in some of the optimization methods. The fact that the optimization method, the assistant method and the distributed capabilities use the same interface will let us compose different types of interactions between them to achieve different results, through a composition of strategies and methods.

In order for the different classes to be distributed, they have to be codified and decodified. Each class that is distributed codifies and decodifies itself.Depending on the codification used, this may greatly optimize the transfer of information since only the necessary details of each of the objects would be transmitted. The way to codify and decodify each object is implemented using the basic types used in c++. Each codification is made using a Packet object which will later be distributed. The Packet class lets us hide the distributed technology used to codify the objects. This is done in order to separate the distributed technology from the rest of the framework (Figure 4). In this way, reusing the codification of the objects when changing the distributed technology would conclude in no modification to the serializable classes. In our default implementation of the framework there exist the MPIPacket class that implements its methods.

As an example for the codification, each bound restriction is represented by an objective (in this case as an integer) and a lower and an upper bound. Therefore, each time a BoundRestriction is sent, we would only have to send one integer and two double values. This should greatly optimize data transfers since only the necessary data is sent.

Finally, in order to use the framework on a grid environment, we use MPICH-G2 [13], a grid-enabled implementation of MPI which uses the grid services provided by the Globus Toolkit. In this way, we can execute our framework in different clusters with a wide range of heterogeneus pro-

Figure 1. Architectural view using the distributed implementation of MPI

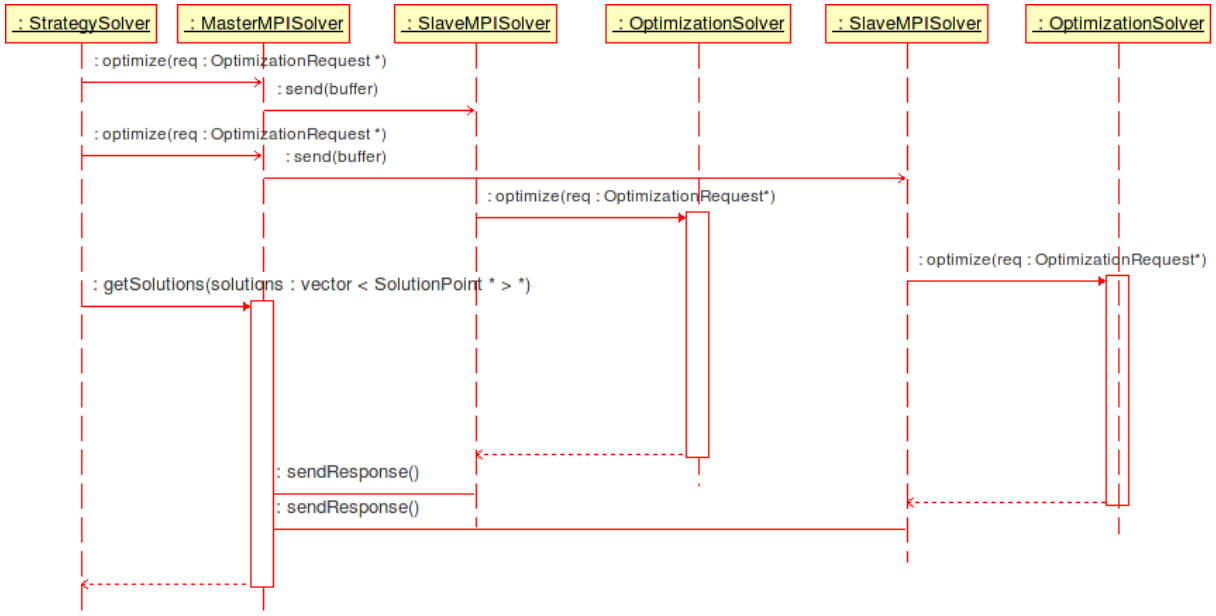


Figure 2. Sequence Diagram showing the distribution of requests

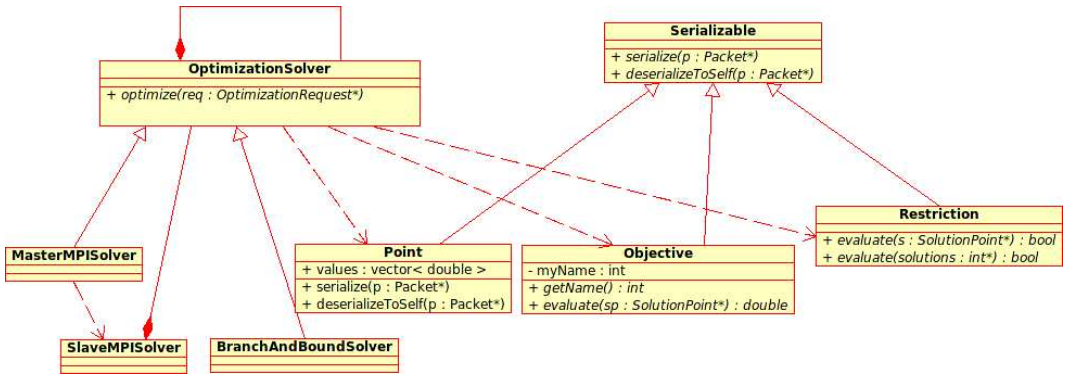


Figure 3. Classes used during the optimization process

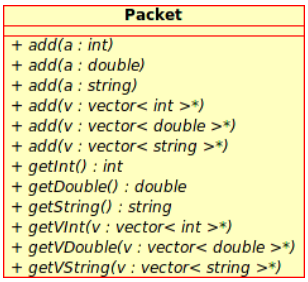


Figure 4. Packet class used for the codification

cessors.

3 Application to the Bi-objective Flow-Shop Problem

3.1 Flow-Shop problem formulation

The Flow-Shop problem is one of the numerous scheduling problems [3]. It has been widely studied in the literature. The problem consists in scheduling n jobs (i = 1 .. n) on m machines (j = 1 .. m). In this paper, we focus on the permutation Flow-Shop where the jobs are scheduled in the same order in all the machines. The two considered objectives are the makespan (Cmax) and the total tardiness (T). The makespan is the completion time of the last job and the total tardiness is the sum of the tardiness of every job. In the Graham *et al.* notation [5] this problem is denoted F/Permut, di/(Cmax,T).
The makespan minimization problem has been proved to be strongly NP-hard by Garey, Johnson and Sehti [11] for permutation Flow-Shops with more than two machines whereas the total tardiness minimization problem has been proved to be NP-hard by Du and Leung [6] even on a single machine.

3.2 The framework instantiation on the problem using the PPM strategy

The PPM strategy consists of three phases which are represented in Figure 5. In this example, the first phase finds the extremes of the two objectives. During the second phase, the search space is uniformly divided with respect to one extreme. Each subspace is optimized with respect to the other extreme. Then, we have a set of uniformly located solutions. The third phase finds the remaining Pareto solutions in all the search space. Using the solutions previously found, as shown in phase three of Figure 5, the space of the multiobjective problem is reduced. The

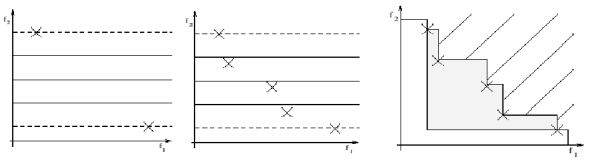


Figure 5. PPM Phases for a bi-objective problem

complete explanation and rationale behind the different steps of PPM can be found in [9].

In order to use the framework for this problem and this method of optimization, one has to implement the classes and methods associated to the strategy, the restrictions used, the assistant method and the objectives of the problem. The classes needed for the instantiation are:

- PPMSolver: It represents the PPM method. In the method solve(OptimizationRequest) we implement the three phases described above. Inside each of these phases, the optimization method is called, for example, to solve one extreme.

- FlowshopObjective: It represents both objectives Tardiness and Makespan. We define the method solve(Solution) which, depending on the kind of objective will evaluate such objective with the solution given. Besides implementing the Objective interface it has to implement bound_abstract. This interface is used by the branch and bound each time it calculates the bounds.

- BoundRestriction: It defines constraints for the 2nd stage of PPM, which several searches are sent in a different search space. Here we define a lower bound and an upper bound and the method that evaluates the solutions between those bounds. The objects of this class will be used by the branch and bound before inserting them into the Pareto front.

- main function: In this function we initialize the framework, create the different classes and we configure them (composition) to work together. That is, we configure the new PPMMSolver with the MasterMPISolver as the internal solver and the branch and bound solver as the internal solver of the SlaveMPISolver.

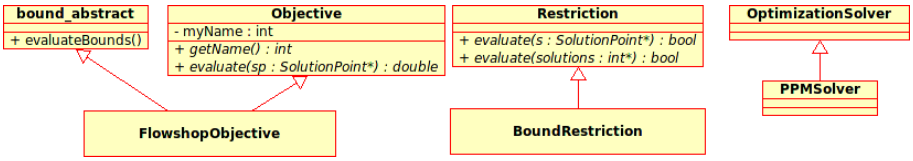


Figure 6. Instantiation of the framework with PPM and the Flow-Shop problem

3.3 Experimentation on Grid5000

Experiments were conducted on the Grid5000 grid. Grid5000 is a nation-wide experimental grid composed by 9 clusters distributed over several French universities (Bordeaux, Lille, Rennes, Sophia-Antipolis, Toulouse, Orsay, Lyon, Grenoble and Nancy). We used the cluster hosted at the Rennes site for the experiments. All of the machines are bi-processors.

Table 1 shows results from different flowshop instances. Inside we show the number of processors used during the different runs, the comparison of the phases of PPM, the total amount of time taken for the resolution, the speed-up of each experiment and the parallel efficiency. We have performed experiments using the instances of Taillard and Reeves. The results show a speed-up that is almost linear with respect to the number of processors. Hence, we think the scalability requirement is met. However, more experiments have to be conducted on different instances and different problems in order to support our initial results.

4 Conclusions and future work

In this paper we have presented an overview of an object-oriented framework for mono-objective and multi-objective exact optimization. As far as we know, there are no frameworks that address the issues of optimizing using a strategy above other methods. We have seen the architecture of the system and the process used for combining the main method or strategy with the assistant method. The codification of the units provided shows a way of codifiyng the objects prior to distributing them taking only the necessary data needed for their identification. Its object oriented nature lets us change the distributed technology without changing the objects codified.
An application using the PPM strategy and a flowshop problem was presented. Through this example we showed the reusability of the framework by implementing only the parts that are needed for the optimization and leaving the rest (like the distribution of requests) for the framework to do. Through numerous experiments with this instance we have proven the scalability requirement is met on a different number of processors.

However, we think that we still have to face different concerns. Specifically, the framework has to be tested with other kinds of problems (TSP, QAP, Knackspack) and other kinds of strategies (K-PPM, TPM, e-contrained). Another issue which needs to be addressed due to the volatile nature of the grid, is fault tolerance. We think this problem could be overcome with different versions of MPI implementations like OpenMPI, FTMPI, MPICH-V2, but we still have to test them with the framework working on the grid. It would be also interesting to integrate this framework with software for heuristic or meta-heuristic optimization.As the interface of the optimization method accepts initial solutions, at least a high level of cooperation is feasible. That is, an heuristic can be used to feed the exact method with initial results. This cooperation will allow the framework to be used for solving larger and more difficult problems.

References

[1] K. Fukuda A. Brunnger, A. Marzetta and J. Nievergelt. The parallel search bench zram and its applications. *Annals of Operations Research*, 90, 1999.

[2] V. D. Cung A. Djerra, B. Le Cun and C. Roucairol. Bob++: Framework for solving optimization problems with branch and bound methods. In *High Performance Distributed Computing*, 2006.

[3] S. Heragu A. Nagar, J. Haddock. Multiple and bicriteria scheduling: A literature survey. *European Journal of Operational Research*, 81:88–104, 1995.

[4] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, J. González, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. Mallba: A library of skeletons for combinatorial optimization. In B. Monien and R. Feldman, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 927–932. Springer-Verlag, Berlin Heidelberg, 2002.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey.

NProcs	Instance	Phase 1	Phase 2	Phase 3	Total	Speed-up	Par. eff.
1	Ta 20-10-1	98min 37s	11m41s	509m23s	619m41s	1	1
60	Ta 20-10-1	3min 6s	45s	9min 24s	13min 43s	45.18	0.75
90	Ta 20-10-1	2min 27s	37s	5min 51s	10min 37s	58.35	0.65
180	Ta 20-10-1	1min 49s	31s	3min 57s	6m 20s	102.71	0.57
1	Ta 20-10-2	159m 41s	23m49s	487m36s	671m8s	1	1
60	Ta 20-10-2	6m 34s	1m30s	14m25s	22m37s	29.67	0.49
90	Ta 20-10-2	4m37s	1m33s	10m13s	16m31s	40.63	0.45
180	Ta 20-10-2	2m57s	1m9s	5m9s	9m20s	74.29	0.41
1	Ta 20-10-3	336m32s	17m51s	442m	795m40s	1	1
60	Ta 20-10-3	6m9s	1m15s	11m17s	19m11s	41.45	0.69
90	Ta 20-10-3	4m32s	51s	8m20s	13m45s	57.82	0.64
180	Ta 20-10-3	2m39s	40s	4m24s	7m47s	102.15	0.57
1	Reeves Easy 20-15-13	104m28s	60m15s	63m40s	228m25s	1	1
60	Reeves Easy 20-15-13	6m32s	2m36s	1m49s	11m14s	20.33	0.34
90	Reeves Easy 20-15-13	5m50s	2m5s	1m33s	9m37s	23.75	0.26
180	Reeves Easy 20-15-13	4m11s	1m24s	1m5s	6m47s	33.67	0.19
1	Reeves Easy 20-15-17	305m29s	51m17s	293m25s	650m12s	1	1
60	Reeves Easy 20-15-17	6m13s	3m37s	5m22s	15m47s	41.20	0.68
90	Reeves Easy 20-15-17	6m39s	1m6s	3m56s	12m5s	53.80	0.60
180	Reeves Easy 20-15-17	4m35s	47s	2m21s	7m54s	82.30	0.46

Table 1. Initial results with PPM and a bi-objective Flowshop Problem

In *Annals of Discrete Mathematics*, volume 5, pages 287–326. 1979.

[6] Du J. and Leung J. Y.-T. Minimizing Total Tardiness on One Machine is NP-hard. *Mathematics of operations research*, 15:483–495, 1990.

[7] C.A. Phillips J. Eckstein and W.E. Hart. Pico: An object oriented framework for parallel branch-and-bound. Technical report, RUTCOR Research Report, 2000.

[8] J. Linderoth J. Goux and M. Yoder. Metacomputing and the master-worker paradigm. Technical report, Angcnne National Laboratory, 1999.

[9] C. Dhaenens J. Lemesre and E-G Talbi. Parallel partitioning method(ppm): A new exact method to solve multi-objective combinatorial problems. *Comput. Op. Res.*, 2006.

[10] Ralph E. Johnson. Documenting frameworks using patterns. In Andreas Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 27, pages 63–72, New York, NY, 1992. ACM Press.

[11] R. Sethi M. Garey, D. Johnson. The complexity of flowshop and jobshop scheduling. *Mathematics of Operational Research*, 1:117–129, 1976.

[12] M. Mezmaz, N. Melab, and E-G. Talbi. A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems. In 21^{th} *IEEE Intl. Parallel and Distributed Processing Symp., Long Beach, California*, Mar. 26-30 2007.

[13] B. Toonen N. Karonis and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63:551–563, May 2003.

[14] T.K. Ralphs and M. Guzelsoy. The symphony callable library for mixed-integer programming. In *The Proceedings of the Ninth INFORMS Computing Society Conference*, 2005.

[15] B. Le Cun T.G. Crainic and C. Roucairol. *Parallel Combinatorial Optimization*, chapter Parallel Branch-and-Bound algorithms. Wiley, John & Sons incorporated, 2006.

[16] S. Tschoke and T. Polzer. Portable branch and bound library user manual library version 2.0. Technical report, University of Padeborn, 1998.

[17] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20:148–165, 1995.

[18] L. Ladson Y. Haimes and D. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on System, Man and Cybernetic*, 1:296–297, 1971.

[19] M. Higaki Y. Shinano and R. Hirabayashi. A generalized utility for parallel branch and bound algorithms. In *Symposium on Parallel and Distributed Processing*, pages 858–865, 1995.

[20] L. Ladanyi Y. Xu, T.K. Ralphs and M.J.Saltzman. Alps: A framework for implementing parallel search algorithms. In *The Proceedings of the Ninth INFORMS Computing Society Conference*, 2005.

Special Session on High Performance Information Retrieval and Visualization: Algorithms and Applications

AN EFFICIENT METHOD FOR COMPRESSING AND SEARCHING GENOMIC DATABASES

Jeffrey B. Wallace, Gregory L. Vert, Sara Nasser
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, NV 89557
E-mail: jwallace@tmcc.edu, {gvert,sara}@cse.unr.edu

KEYWORDS
Genomic sequence, search, algorithm, compression, bioinformatics, database

ABSTRACT

Biological databases are growing significantly, as are the number of queries directed at them. In 2005, the genomic databases at the National Center for Biotechnology Information (NCBI) received about 50 million web hits per day, at peak rates of about 1,900 hits per second. As these databases become more popular, there is increased demand to make them faster and more efficient. In this paper, we propose a method for compressing and searching selected genome databases using techniques appropriate for computers of virtually any size. This search technique is expected to produce its best results with large search sequences against large DNA databases, and lends itself to parallel computation techniques with little communication overhead required. Because the compression algorithm uses a lossless binary encoding format, search results are exact – not approximate. Furthermore, searches take place on the compressed data, obviating the need for decompression prior to executing a search.

INTRODUCTION

Biological databases are growing significantly as organisms are being sequenced. These genomic databases help biologists understand the underlying structure of organisms and aid research in the area of genomic sciences. Publicly available databases can be used by biologists to compare organisms, find related species, etc.

To retrieve information, users run queries against one of these databases, such as a search for a nucleotide sequence from a nucleotide database. A typical search involves matching a query nucleotide sequence with all the sequences available in the database. If a database is large in size, several comparisons are required until a match is found. Efficiency is critical in such a database system. The user desires to find the exact or most similar result in the shortest amount of time. A database search can be viewed as finding the longest common subsequence(s) between the query sequence and a database sequence. A longest common subsequence indicates the similarity between the query and the database sequence.

Genomic sequences are fairly large in size ranging from several thousand to million character long sequences. Searching against a large database can be time consuming therefore there is a need to make database access faster and better.

A typical search may involve comparing a string of 200K characters against a database that may contain millions of sequences of similar or larger sizes. A user who may be trying to search against a large database may require several minutes to get the reply. Added to that a PC has limited resources and searching against a large database can be quite time consuming. Querying a database involves several factors such as the speed of the tool, accuracy of the match, etc. An ideal tool for such querying should satisfy these requisites.

Since large databases have huge memory requirements, a method to compress data can be beneficial. Compression of data can greatly reduce the processing time. For example, if the data is compressed five-fold, then a 200,000 character sequence now becomes a 40,000 character sequence. The same query search now involves comparison with smaller sequences making the search faster.

DNA data is sensitive to changes, such as replacements, insertions, deletions. A compression technique that permits full recovery of the genome sequences is required. For example, consider two sequences:

Sequence 1:

ACTTACGTATCGCCCCC

Sequence 2:

ACTTACGTATCGCCACC

Sequence 1 and 2 are similar in that there is only one character difference between them. An ideal compression technique should maintain the similarity relationship between the two sequences after compression. In the above case the compressed data should also have a distance of 1.

The research in this paper focuses on providing fast search, fast retrieval speed from disk, efficient memory utilization, and easily parallelizable implementation for even faster searches and/or distributed deployment. A

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

lossless compression technique is proposed that allows full recovery of data. The next section presents the background, followed by the proposed algorithm in the following section. In the following section we analyze the technique and conclude with future work in the last section.

BACKGROUND

In this section we discuss some of the existing tools for sequence alignment. We also discuss some of the earlier methods propose to improve sequence alignment.

Sequence alignment is an important field in bioinformatics. Sequence alignment provides method to compare new sequences with previously completed or assemble sequences. The completed sequences are stored in databases.

Genomic database servers process tens of thousands of queries a day. GenBank is one such database maintained by National Center for Biotechnology Information (NCBI). GenBank has over 55 million sequence entries from at least 200,000 different organisms (GenBank 2005). GenBank's search tool is known as BLAST. In 2005, NCBI received about 50 million web hits per day, at peak rates of about 1,900 hits per second, and about 400,000 BLAST searches per day from about 2.5 million users (Astell 2005).

BLAST is a de facto standard tool used for measuring similarity between sequences (Altschul 1990, 1997). BLAST is popular for its efficiency, and has undergone several updates for efficiency and speed. Mega BLAST is a greedy search method that works on BLAST data for DNA sequence alignment search and is known to be faster than BLAST. Mega BLAST uses a greedy algorithm for nucleotide sequence alignment search (Zhang 2000). This program is optimized for aligning sequences that differ slightly and can align much longer sequences than BLAST. Mega BLAST can only work with DNA sequences.

The Institute for Genomic Research (TIGR) is another center for deciphering and analyzing genomes. TIGR's Genome Project contains a collection of curated databases containing DNA and protein sequence, gene expression, cellular role, protein family, and taxonomic data for microbes, plants and humans (IGR 2007).

BLAST is a tool that does an exhaustive search. FASTA is another tool that does an exhaustive search. An exhaustive search is costly in terms of speed. A query string does not have to be compared to the entire database. Heuristics can be added to a search process to make it faster and accurate. Keyword based searches are one such example of non-exhaustive search. Keyword-based search has been popularized by Internet search engines and is not generally provided by traditional databases (Agrawal 2002). An example of keyword-based search over structured databases is EKSO (Su 2005). EKSO indexes interconnected textual content in relational databases, providing intuitive and highly efficient keyword search capabilities over this content. It trades storage space and offline indexing time to significantly reduce query time computation.

There are several other search tools such as Flash, SST and CAFÉ (Cao 2005, Baxevanis 2005). CAFÉ is based on an indexed scheme with appropriate data structures and has shown a faster query search than exhaustive searching. A comparison of BLAST, FASTA, and CAFÉ has been studied (Williams 2002). An indexing technique for answering approximate keyword search queries was developed by Fei and Mefford (Shi 2005). This technique has two principal components – a data structure called V-tree and its partition methods for clustering words in the vocabulary into subgroups. It stores the words in the vocabulary into a V-tree based on its partition methods. The V-tree data structure can reduce the number of distance computations needed to answer the query.

Even though there has been much research for making for making query searches faster, there has been less research in terms of reducing storage requirements for both the database and the user. The algorithm proposed in the following section can perform searches on compressed data in a fast and efficient manner.

THE PROPOSED METHOD

This section describes the proposed compression and searching algorithms, including sequence encoding, data structures, and query processing. Examples are provided in each of these areas.

Sequence Encoding and Compression

Genomic sequences are commonly stored as strings comprised of the four DNA bases: C, G, A, and T. Generally, these are stored as ASCII characters, where each symbol requires a single byte of storage.

However, only two bits are required to adequately express these four symbols. Thus, 'C' can be replaced with the bits 00, 'G' with 01, 'A' with 10, and 'T' with 11. This simple binary encoding of base strings allows four DNA bases to be represented within one byte. Thus, the sequence AGGT can be represented in binary form as 10 01 01 11, which is readily converted to its decimal equivalent of 151.

For the purposes of the proposed search algorithm, however, it is desirable to represent octets of DNA bases as a single unit. Since each base requires two bits, all possible octets can be represented as 16-bit integers ranging from 0 (all Cs) to 65535 (all Ts) as shown in Figure 1. Since each integer in this range uniquely

encodes a string of eight nucleotides (an octet), these integers will serve as hash values in the data structure at runtime. In the meantime, these compressed values are either stored sequentially on disk or transmitted sequentially over a network during a file read operation.

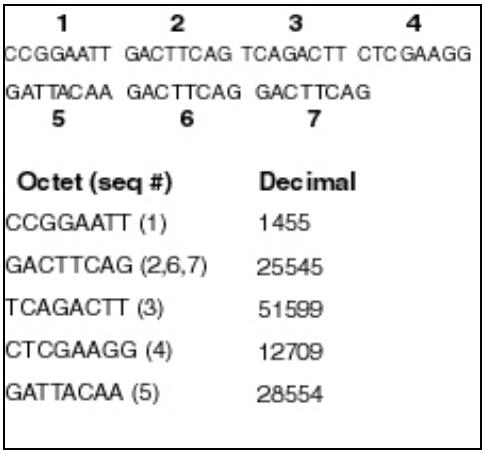


Figure 1: Octet Encoding

Data Structure

Since DNA search and target strings are typically very large, simple M x N string comparisons take too much time to be considered a viable solution – a more sophisticated data structure is required in order to achieve reasonable performance in large searches. The applicability of a data structure is largely determined by the context of the problem – there is no such thing as the "perfect" data structure. In the context of genomic search, tree structures suffer because the data has little order other than sequence, so complete searches require either complete tree traversal or extra links within the tree structure. Arrays suffer from the same problem. Various array indexing schemes offer speed improvements, but at the expense of memory. The technique used for video compression suffers because the distances to reference words are typically longer than the two bits necessary to store the base being encoded (1).

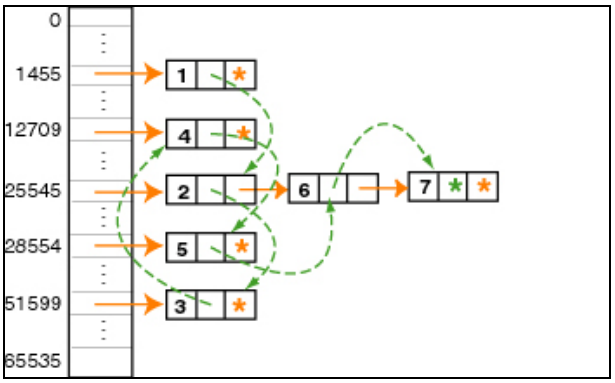


Figure 2: Data Structure

The proposed data structure consists of a doubly-chained hash table, where one chain of pointers (solid lines in Figure 2) implements a linked-list of all octets that hash to the same integer value. A second linked list (shown with dashed lines) is used to traverse the octets in sequential order. In addition to preserving the integrity of the data, this structure ensures that only the portions of the targets sequence that match the search string will be searched.

This data structure is easily serialized. Each octet is encoded to binary (interpreted as decimal) form as described above and stored sequentially on disk – which offers the best I/O performance at runtime.

As each octet (16-bit integer) is read, a new node is created and inserted at the end of the linked list for the octet's hash value (solid lines), which ensures that each hash value's linked list is stored in sorted order. The node also contains sequence position information in order to report where a match begins, as well as a second linked list (dashed lines) that allows the data structure to be efficiently traversed sequentially. Both of these linked lists are traversed during a search operation.

```
startNode = hashTable( decimal value of first
octet of search string )

FOR EACH of the possible ((# bases mod 8) + 1)
search windows, DO
    searchNode = first complete octet of the
    windowed search string

    WHILE startNode != null, DO
        currNode = startNode

        WHILE currNode.hashValue ==
        searchNode.hashValue

            currNode = currNode.next
            searchNode = searchNode.next

            //end of target without match?
            IF currNode == null AND searchNode != null
                BREAK loop

            ELSE IF searchNode == null OR
            searchNode is a partial octet
                IF searchNode.partialBits !=
                currNode.partialBits
                    BREAK loop

                ELSE report MATCH at
                startNode.sequencePos*8 + windowNum

        startNode = startNode.next
```

Figure 3: Algorithm

Algorithm

Sequence searching takes place on compressed sequence octets stored in the data structure described above. In order to make comparisons to this data structure, the search string also needs to be encoded as binary octets using the same compression technique. Because of the iterative windowing required by this algorithm, search strings need to contain at least fifteen bases.

Descriptions and examples of each of the algorithm functions are covered below. The algorithm shown in Figure 3 assumes that the target sequence has been encoded and stored in the data structure described above. It also assumes that each base search sequence has been encoded as a two-bit binary value, and that the search sequence contains at least 15 bases. In the following algorithm, the variable currNode follows the sequence-ordered linked list (the dashed list in Figure 2). The variable startNode follows the hash-ordered linked list (the solid-line list).

Fully Aligned Octets – A Simple Example

Figure 4 builds on the previous figures, and shows a simple, near-best case scenario. In this case, the search sequence matches the 6th and 7th octet of the target sequence.

The search begins by accessing the linked-list addressed by the hash value of the search sequence's first octet. In this case, the first element of the hash value's linked list is octet #2 in the sample sequence. Indeed, the first octet of the search sequence matches octet #2 of the sample sequence, so the (dashed-line) linked list is followed to find the octet #3 in the sample sequence. In this case, the hash value of octet #3 of the sample sequence (51599) does not match the hash value of the second octet of the search sequence (25545), so this once-promising search is abandoned, and the (solid-line) linked list is followed to the next octet that matches the first octet of the search sequence – in this case, octet #6.

The next possible match begins at octet #6 in the sample sequence. Following the (dashed) linked list to octet #7 reveals that it matches the second octet in the search sequence. Since there are no more terms in the search sequence, there is a match beginning at octet #6 in the sample sequence. If there were more octets in the (solid-line) linked list, further matches could be discovered by repeating the process until the end of the (solid-line) list is reached.

Partial Search Strings

In the previous example, the lengths of the search strings were multiples of eight, so they aligned nicely with octet boundaries. More often than not, this perfect alignment is not the case in actual searches, so the algorithm requires a further refinement in order deal with non-aligned search strings. Instead of the last octet of the search sequence being compared to an entire octet in the target sequence, partial octets are compared using an AND operation as shown in Figure 5. As will be seen in the next section, this partial alignment technique may also be used on the first octet of the search string.

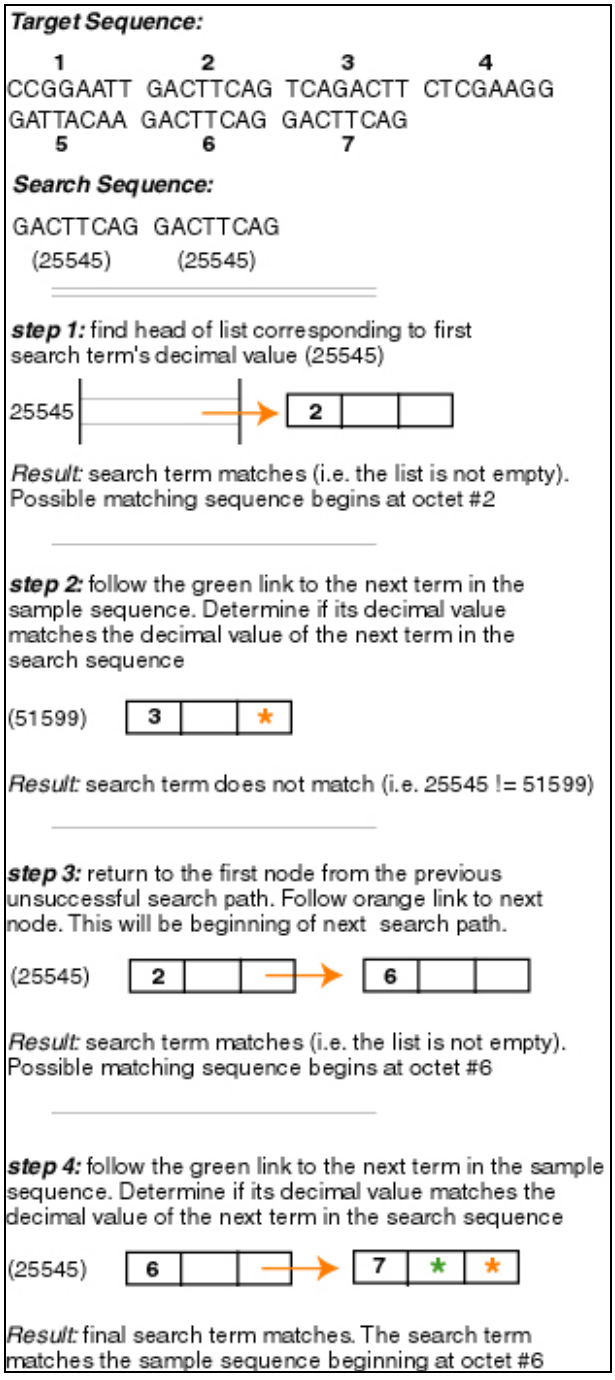


Figure 4: Searching the Data Structure

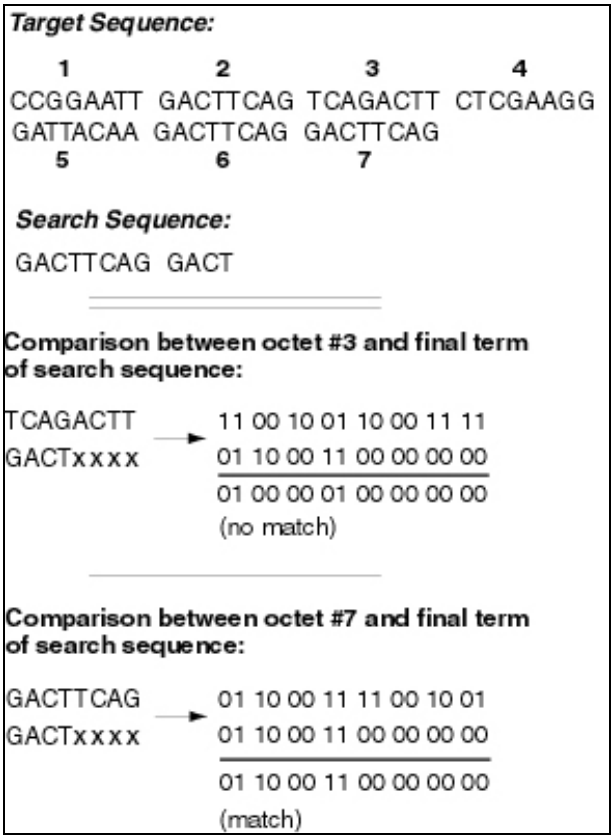


Figure 5: Partial Octet Matching

Search Sequence Windowing

In both of the examples described above, the matching sequences occurred on octet boundaries. Again, this is not usually the case in actual searches. In most cases (or when searching for all matches in a target string), the search algorithm uses a windowing technique where the search sequence is shifted one base (2 bits) to the right for each of the eight possible positions in the octet. Note that only the search sequence is shifted – the data structure for the target sequence remains unchanged. Although this requires eight separate searches of the target sequence, these basic comparison operations are sufficiently fast in modern processors.

An example of the windowing algorithm is shown in Figure 6. At each iteration, the octet contained in the box would serve as the first octet to be matched. As before, the hash value is simply a decimal interpretation of the binary base encoding.

For each window iteration, the algorithm begins by matching all complete octets as illustrated previously in Figure 4. As usual, if any mismatches occur along the way, the search moves to the next target sequence matching the hash value of the first octet of the windowed search string. If all complete octets match, the partial octet at the end of the search sequence (if it exists) is matched against the target sequence as illustrated previously in Figure 5. If the sequences continue to match, the partial octet at the beginning of the search sequence (if it exists) is finally matched against the target sequence. Note that this data structure does not include backward chaining – which would require unacceptable memory overhead because it would require extra pointers for every octet, even though only one backward pointer would be used for a given search string. Instead, the partial sequence preceding the first complete octet is retrieved from disk, which is a relatively fast direct access file I/O operation since the sequence position is known.

Figure 6: Search Windowing

ANALYSIS

A software prototype that implements this method has been developed. Preliminary testing on a dataset containing 5.6 million base pairs appears promising:

a) The simple compression algorithm achieves a constant 4:1 compression ratio on standard ascii-coded FASTA databases.

b) Base octets appear to be reasonably well distributed, resulting in similar-sized linked lists for each possible hash value. Thus, in a target sequence with 3 billion base pairs, the average linked list for

a given hash value (the solid-line list) would be expected to contain approximately 5700 nodes (3B $\div 2^{16} \div 8$).

c) Searches for strings containing up to 5000 base pairs are executing in under 100 ms on an (old) 2GHz AMD 64 with 1 MB of RAM.

CONCLUSIONS AND FUTURE WORK

The expected benefits of this approach are four fold:

a) Fast search. Only the portion(s) of the target sequence that match the search string will be searched. Non-matching areas are ignored.

b) Fast I/O. Compressed data are saved sequentially, resulting in fast retrieval speed from disk.

c) Efficient memory utilization. Compressed data does not need to be decompressed. The algorithm works directly on compressed data.

d) Easily parallelizable for even faster searches and/or distributed deployment. The algorithm is embarrassingly parallel and involves little communication overhead.

As described above, this data structure relies heavily on memory address pointers. Since these pointers cannot be meaningfully stored on disk (when the data is reloaded, it will most likely load into different memory addresses), search operations will be most efficient on machines with enough memory to store the entire compressed target sequence.

The memory required to store a target sequence is expected to be directly proportional to the number of bases in the sequence and the word size of the computer used. With a 64-bit processor, each octet is expected to require 20 bytes (32 bits for a sequence number and two 64-bit addresses). With a 32-bit processor, each octet is expected to require 12 bytes. Thus, the data structure required to hold a target sequence with 3 billion base pairs on a 64-bit processor is expected to require approximately 7.5 GB of memory. Although this is a significant amount of memory, it is within the realm of a workstation-sized computer.

Preliminary testing of this approach appears promising. Over the next couple of months, the following extensions to this project are anticipated:

a) Finish programming the prototype search tool.

b) Measure search speed of various-sized search strings on a broad range of actual genomic datasets, and compare these results against those using tools such as BLAST, FASTA, and CAFÉ.

c) Compare memory and storage efficiency against existing tools such as BLAST, FASTA, and CAFÉ.

d) Parallelize the algorithm so it can take advantage of multiple processors and large shared memory clusters. This is likely to lead to significant performance gains.

e) Explore methods for implementing wildcard (similarity) matching in order to be able to search strings "similar" to a search string.

REFERENCES

Agrawal, Sanjay, Surajit Chaudhuri, Gautam Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases," *18th International Conference on Data Engineering* (ICDE'02), 2002.

Altschul S., W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic Local Alignment Search Tool," J. *Molecular Biology*, vol. 215, pp. 403-410, 1990

Altschul, SF, TL Madden, AA Schaffer, J Zhang, Z Zhang, W Miller, DJ Lipman, (1997) "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Res*, . 25, 3389–3402

Astell, James (2005), *"Databases of Discovery",* ACM Queue vol. 3, no. 3 - April 2005

Baxevanis, Andreas D. (Editor), B. F. Francis Ouellette, *"Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins"*, New Jersey: Wiley 2005

Cao, Xia, Beng Chin Ooi, Tung, A.K.H., Hwee Hwa Pang, Kian-Lee Tan, "DSIM: A Distance-Based Indexing Method for Genomic Sequences", *IEEE Symposium on Bioinformatics and Bioengineering*, 2005. BIBE 2005, Publication Date: 19-21 Oct. 2005, (pp): 97- 104

GenBank (2005), http://www.nlm.nih.gov/news/press_releases/dna_rna_100_gig.html, date accessed Feb 2007.

The Institute for Genomic Research, http://www.tigr.org/db.shtml, data accessed March 2007.

Shi, Fei, Mefford, C., "A new indexing method for approximate search in text databases"**,** *The Fifth International Conference on Computer and Information Technology*, 2005. CIT 2005, Sept. 2005 pp: 70- 76, 2005

Su, Qi, Jennifer Widom, **"**Indexing Relational Database Content Offline for Efficient Keyword-Based Search**"** *Proceedings of the 9th International Database Engineering & Application Symposium* (IDEAS'05), Vol. 00, pp: 297 – 306, 2005

Williams, H. and J. Zobel. Indexing and Retrieval for Genomic Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2002.

Zhang, Zheng, Scott Schwartz, Lukas Wagner, and Webb Miller (2000), "A greedy algorithm for aligning DNA sequences", *J Compuational Biol* 2000; 7(1-2):203-14.

AUTHOR BIOGRAPHIES

JEFF WALLACE was born on a farm in the upper peninsula of Michigan. He received his BS in Computer Science from the University of Michigan in 1982, his MBA from Santa Clara University in 1988 and his MFA in film with a specialization in Special Effects from the University of Southern California in 1996. He is currently a graduate student at the University of Nevada and is a tenured faculty member at Truckee Meadows Community College in the Department of Computer Office Technologies. His

research interests are in Bioinformatics and Artificial Intelligence. His email is jwallace@tmcc.edu

GREGORY VERT was born in Fairfield California in 1956. He received his BS in Geography with a specialization in GIS from the University of Washington in 1985, his MS in Information Systems Management, Seattle Pacific University, in Seattle, Washington in 1988, and his PhD in Computer Science from the University of Idaho in Moscow Idaho in 2000. He has been an Assistant Professor at the University of Nevada, Reno in the Computer Science and Engineering Department since 2002. His research is in the areas of GIS, Computer Security, Fuzzy System, Database and Bioinformatics. His email address is gvert@cse.unr.edu

SARA NASSER was born in Hyderabad, India. She received her BE in Computer Science and Engineering from MJCET, Osmania University, and her MS in Computer Science from the University of Nevada in 2003. She is currently a PhD student in Computer Science and Engineering and is expecting to graduate in 2008. Her research is in the area of Bioinformatics and Fuzzy Systems. Her e-mail address is sara@cse.unr.edu

VOLUMETRIC VISUALIZATION METHODS FOR ATMOSPHERIC MODEL DATA IN AN IMMERSIVE VIRTUAL ENVIRONMENT

Michael P. Dye[1,2]
Philip A. McDonald[2]
Frederick C. Harris Jr.[1,2]
William R Sherman[2]

[1]University of Nevada, Reno
Reno, NV 89557

[2]Desert Research Institute
Reno, NV 89512

mdye@cse.unr.edu
phil.mcdonald@dri.edu
fredh@cse.unr.edu
bill.sherman@dri.edu

KEYWORDS
3D, Scientific Visualization, Surround Screen Virtual Environment

ABSTRACT

We present the design and implementation of an immersive interactive volumetric visualization system. This system was designed to allow atmospheric modelers to visualize their simulations in an immersive environment such as our Fakespace FLEX system. By allowing them to play back the entire simulation as well as choose what airborne particulates to turn on and off atmospheric scientists are given an incredible degree of control regarding what to study and look at from any angle with amazing detail at the interactions between particulates in a simulation and interactions of the particulates with surrounding terrain.

INTRODUCTION

Atmospheric simulation is an important means of understanding the environment around us. Through atmospheric simulation we can predict how various airborne particulates such as dirt, smog, and fire can affect our cities and overall public health. However, problems can arise when trying to interpret the results of the simulation. For example, it can be difficult to determine the density or shape of an individual particulate if the area under study is large. A similar problem arises when attempting to find how various terrain formations interact with and affect the atmosphere. However, little in the way of research exists on how to accurately model atmospheric data at interactive frame rates.

Since raw data can be hard to conceptualize, particularly when the size of the data sets can be hundreds of megabytes if not gigabytes, an alternative method of interpreting atmospheric data is needed. Virtual reality technology allows us to accurately and realistically model atmospheric data in a meaningful way for the user. Virtual reality has long been used as a way of creating realistic visual simulations to help aid in interpreting large and complex data sets. Recent advances in visualization and supporting technologies now offers the possibilities of creating realistic, real-time atmospheric visualizations for research.

By combining virtual reality technology with atmospheric simulation users are able to visually conceptualize large amounts of atmospheric data in an interactive way. This paper describes a package we call Vesuvius which is a virtual reality library for visualizing large sets of atmospheric data. Vesuvius is intended to allow users to visualize various atmospheric data sets from a variety of viewpoints and to allow for playback of atmospheric data sets that contain temporal information.

The rest of this paper is structured as follows. First we present some related work on atmospheric visualization, virtual reality, and previous research on volume rendering. Next we present an overview of the execution environments followed by our immersive volume visualization library for visualizing atmospheric data. Lastly presents our conclusion and possibilities for future work.

BACKGROUND

Atmospheric visualization would benefit from the merger of both volume visualization and virtual reality. Marrying the two, users can see realistic 3-dimensional representations of their atmospheric simulations. In addition, it allows users to immerse themselves in the simulation, enabling them to watch it from different points of view in a highly realistic environment.

Atmospheric Simulation

Atmospheric modeling and simulation is done with numerical models like the National Center for Atmospheric Research (NCAR)/Pennsylvania State University Mesoscale Model (MM5). Models such as MM5 simulate the behavior of a wide range of atmospheric parameters, including mass parameters (e.g. temperature and humidity) as well as momentum parameters (e.g. wind U, V, and W fields). When these models are initialized with archived meteorological data for prior conditions, their simulations can be used to analyze the state of the atmosphere during a past event. When they are initialized with current meteorological observation data, their simulations serve as atmospheric forecasts. These models are quite adaptable and can be utilized for atmospheric simulations over domains with resolutions ranging from tens of kilometers down to just a few kilometers. Ongoing research is further refining the physics in MM5 so that simulations over domains with horizontal

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

resolutions of less than one kilometer can be achieved.

In addition to models like MM5 which simulate the state of the atmosphere, there are specialized models that simulate a specific component of the atmosphere. One such model is the Comprehensive Air Quality Model with Extensions (CAMx) which simulates the behavior of atmospheric chemicals and its use is common in air pollution studies. Like MM5, CAMx can simulate past, current, or future events and can be scaled to adapt to a wide variety of domain sizes and resolutions.

Atmospheric Visualization

In 1988, the Space Science and Engineering Center (SSEC) at the University of Wisconsin, Madison released the Vis5D atmospheric visualization tool and in 1994, a virtual reality port of this open-source application was created for the CAVE immersive display system. Vis5D, with it's grid limitations, displaying smog data over Los Angeles can be seen in Figure 1. Since these early efforts, a great deal of the work done in the area of atmospheric visualization has been in the field of cloud simulation and rendering (Dobashi et al., 2000; Ebert and Parent, 1990; Harris and Lastra, 2001). Because of the level of complexity in rendering clouds (realistic shading, light scattering, etc) much previous work has been done in non-interactive rendering techniques for cloud rendering. Work into interactive cloud rendering has produced methods whereby most scattering and shading are done in a preprocessing step and textured polygons known as imposters are used to bypass fill rate limitations (Harris and Lastra, 2001).

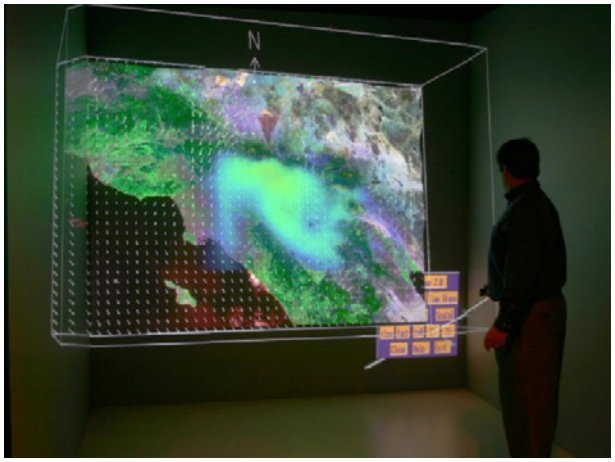


Figure 1. Volume Rendering in Vis5D

Volume Rendering

Volume visualization deals with displaying volumetric data sets, represented as sample points. There are two means of achieving this. First is indirect volume rendering (IVR) which involves converting the volumetric data into a set of polygonal iso-surfaces which are then rendered using traditional graphics hardware. The second means is called direct volume rendering (DVR) which involves rendering directly to the screen without an intermediate step such as converting to iso-surfaces.

Indirect volume rendering assumes that extractable iso-surfaces exist, which is not always the case (such as flow fields, clouds, etc.). In addition, the complexity of the iso-surfaces might be so complex that they may overwhelm the capabilities of the graphics hardware. Because of this, direct volume rendering may be more efficient. What follows is an explanation of the primary methods of direct volume rendering.

Raycasting

Raycasting is perhaps the most researched of direct volume rendering algorithms (Levoy, 1988, 1990) with several acceleration techniques for more interactive volume raycasting being proposed over the last decade (Grimm et al., 2004; Knittel, 2000; Mora et al., 2002; Wan et al., 1999).

For each pixel in the image, a ray is cast into the volume. At fixed intervals along the ray the volume data is resampled, most commonly using tri-linear interpolation. Trilinear interpolation involves taking the scalar values of the eight neighbors to a particular pixel and weighing them according to their distance to the actual location. The solutions to each interval are combined either in a front-to-back or back-to-front order to determine the final color and opacity of the pixel. Because each pixel has to be computed, raycasting is the slowest of volume rendering algorithms. However, because volumes are determined on a pixel-by-pixel basis, they can generate high-quality images without any blurring or loss of detail.

Shear-warp

Shear-warp (Lacroute and Levoy, 1994) is a fast means of software volume rendering. Unlike raycasting, no rays are cast into the volume. Instead, the volume is projected slice by slice onto the image plane using bi-linear interpolation within the slices. In shear-warp, an intermediate image plane is created and aligned with the volume. The volume itself is then sheared to turn the projection direction into a direction perpendicular to the intermediate image plane. This intermediate image is finally warped to the final image plane. Due to the warping to the final image plane only being required once per image, not once per slice and run-length encoding of the volume data, shear-warping is considered the fastest software base volume rendering algorithm. However, the speed of shear-warp does not come without a price, magnification of volume data results in blurring of details and the addition of artifacts. In addition, it requires three copies of the volume data to remain in memory, one copy for each major axis.

Texture-Based

3D texture mapping uses 3D textures to render volumes. The volume is loaded into texture memory and sampled as a series of slices. The resulting planes are then drawn as a series of textured polygons that are blended together, thus creating the final image. Because texture mapping and compositing are performed in hardware, the rendering is actually faster than shear-warping for small datasets. However, if a dataset if too large to fit completely into texture memory,

then performance is decreased considerably as data must continually be paged back and forth between the hard drive and graphical memory. To minimize the performance impact of this, the volume is encoded into an octree structure (Boada et al., 2001; Fang et al., 1996).

Virtual Reality

VR provides a medium composed of immersive interactive computer simulations that provide real-time feedback to the users (Sherman and Craig, 2003). VR is a technology that can provide sophisticated real time 3D user interface for users to interact with 3D applications. Therefore, VR technology is a good candidate as a user interface for interaction with 3D atmospheric data. There have been various research efforts regarding the use of VR technology in different contexts (Chen et al., 2001; Koller et al., 1995; Lin et al., 2000; Loftin et al., 1998).

Interaction within a virtual environment can generally be categorized into selection, manipulation, navigation, and system control. Selection and manipulation techniques will be very different from one application to another as each application's interaction requirements are different. Navigation techniques are mandatory for large scaled VR applications like terrain visualization since the user will need to be able to get from one point to another within the virtual environment. In system control interaction, the user will be able to control the state of the application at the system level where the execution mode of the application is changed.

The research of applying VR technology into the field of atmospheric research is a little explored research direction. The capability to see various types of volumetric data in a 3D large screen immersive environment has not been explored. In the research of large displays, (Huang et al., 2006) described some of the unique features provided by their use. These unique features provided by large displays could be beneficial in the viewing and interaction with large atmospheric datasets within a virtual environment.

HARDWARE AND SOFTWARE ENVIRONMENT

Our immersive visualization facility hardware includes both a four-screen CAVE-like Fakespace FLEX display (Figure 2), and a single-screen Visbox-P1 (Figure 3). The FLEX display is driven by an SGI Prism running SuSE 9.0 Enterprise edition, with four active-stereo capable graphics channels. Tracking of the participants is accomplished with an Intersense IS-900 VETracker with wireless Mini-Trax Head Tracker and wireless Wand with 5 Buttons and center click joystick. Both the viewpoint and the dominant hand are tracked at interactive rates to enable participants to interact with the application

For the VisBox-P1 virtual environment system, a custom built dual Opteron graphics workstation running OpenSUSE 10.0 with an NVIDIA GeForce 6800 GT with dual outputs is used to drive two projectors. Tracking of the participants is accomplished with an Ascension Technology Flock of Birds with one sensor for positional tracking of the gamepad used for button and joystick inputs. The tracked gamepad is used by the participants to interact with the application. The viewpoint of the participant's head is tracked using proprietary optical tracking solution.

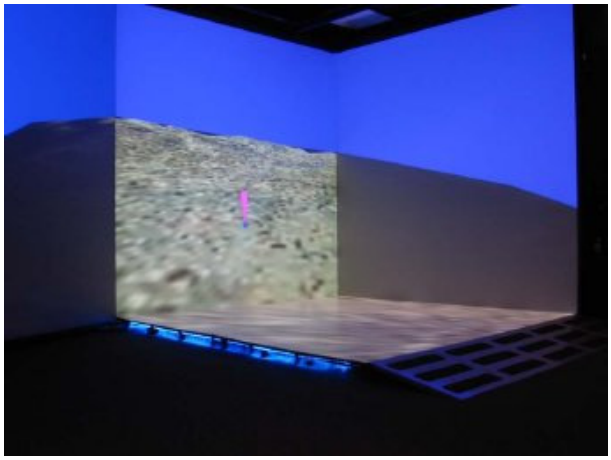


Figure 2. FakeSpace FLEX Display

Figure 3. VisBox-P1 Display

Vesuvius is designed to work well inside the open-source FreeVR (Sherman, 2007) and OpenSceneGraph (OSG) libraries. The FreeVR virtual reality integration library is a cross-display VR library with built-in interfaces for many input and output devices. It allows programmers to develop on a standard desktop machine, with inputs and display windows that simulate a projection or head- based immersive system. The application can then run on either the Visbox-P1 or FLEX displays, or the display of a collaborator on just about any type of VR system. The OpenSceneGraph library is used to help with world rendering. OSG allows 3D objects

to be hierarchically organized within the environment, and also provides a system that optimizes the rendering through the use of various culling and sorting techniques.

Although we chose the VisBox-P1 and the FLEX as the display devices, the application can easily be modified to display on different virtual environment displays with various configurations supported by the FreeVR library. In addition, the FreeVR library also supports various virtual reality devices that enable the use of other tracking devices with the application.

PROPOSED ARCHITECTURE

Although much has been done in the realm of volume visualization little has been done in atmospheric visualization, specifically with the MM5 model. Most volume visualization research has focused on visualizing static medical data such as CAT scans and MRIs. Unlike medical imagery, there is often a temporal variable associated with atmospheric data sets. This means that for a given particulate, the user might wish to see its simulated interaction over the course of minutes, hours, or even days. In addition, the size of atmospheric data tends to be considerably more than that of medical data if for no other reason than the scope of the datasets. Work in this area of dynamic volumes is far from complete. In addition, atmospheric data contains many types of divergent data, each of which may or may not be wished to be displayed by the user. The purpose of Vesuvius is to address these differences by visualizing dynamic and varied atmospheric data at interactive framerates.

Design

At the core of Vesuvius are two direct volume rendering algorithms: raycasting and shear-warping. Vesuvius was designed with the algorithms derived from a common base to create a minimum amount of code duplication and increase overall program cohesion. As an added result of this derived structure, other volume rendering algorithms such as texture-based volume rendering can be easily added in the future. In addition, this allows for a common interface through which the algorithms can receive data (such as camera position) and send data (such as the final volumetric image). An overview of this design is given in Figure 5.

Raycasting was chosen as one of the direct volume rendering algorithms to implement because of its ability to display highly detailed volumes. Unlike other algorithms such as shear-warping, increasing the resolution of a volume or its magnification will not result in any loss of detail when using the raycasting algorithm. However, due to the computationally intensive nature of the algorithm, even with optimizations such as early ray termination (Levoy, 1990), spatial encoding of the volumetric data (Grimm et al., 2004; Sobierajski and Avila, 1995), or an object-ordered approach (Mora et al., 2002) the framerate could barely be considered interactive, especially in a virtual environment. This algorithm is particularly suited to static, highly detailed volumetric data such as that of medical data which comes from CT and MRI scanners.

The second algorithm chosen was shear-warping because of how it complements raycasting. While shear-warping tends to blur detail when the resolution is increased or the volume is magnified, under normal viewing conditions it is the fastest of the software volume rendering algorithms. The speed of shear-warp allows the amount and size of information normally contained in volumetric data to run at interactive framerates. This is made more important when temporal data is factored into the volume data, where the volume will be required to dynamically move and change in real-time. With the amount of data being displayed in such a way, it becomes essential that things run at interactive framerates.

Once the image is obtained, a texture is created and placed on a billboarded polygon at the position of the actual volume data. This minimizes the amount of data that actually needs to be rendered as well as minimizes the amount of calculating that needs to be done, as the polygon texture only needs to be recalculated when the camera position changes.

Vesuvius works with an external program that will read and convert MM5 files into a custom file format. Because of the sheer size and complexity of MM5 files it is difficult to find and extract just the relevant volumetric information for a given frame in real-time. This external program extracts all relevant volume information (including bounds, opacities, etc.) for each particulate and stores this into the custom file format. In cases when there is temporal data in the MM5, the step is recorded and the temporal data is stored as deltas at each step interval. A command line option in the converter program allows the user to specify a resolution for the time interval. This means that the user can specify a higher resolution for the temporal data stored in the MM5 file and the program will interpolate the temporal data and add this new information into the file. While this greatly increases the file size, it also makes animations of atmospheric data both smoother and less computationally intensive, which results in greater framerates. The layout of this file structure is given in Figure 4.

What makes Vesuvius unique is the amount of control it gives the user when viewing the atmospheric data. It encapsulates controls for moving through the temporal data in the dataset, allowing the user to rewind, fast-forward, play, and pause atmospheric simulations in real-time. In addition, Vesuvius can render overlapping volume data such as that of multiple particulates. This is done by first keeping track of each particulate separately and computing how much each particulate adds to the overall volume data at a given point. From this, the user can choose to display, hide, or adjust the transparency for any given particulate type. This allows the user to focus on the interactions of one or several certain particulates with the atmosphere and for the user to see how these particulates are affected by the varying conditions of the simulation without the added information of every other particulate in the dataset.

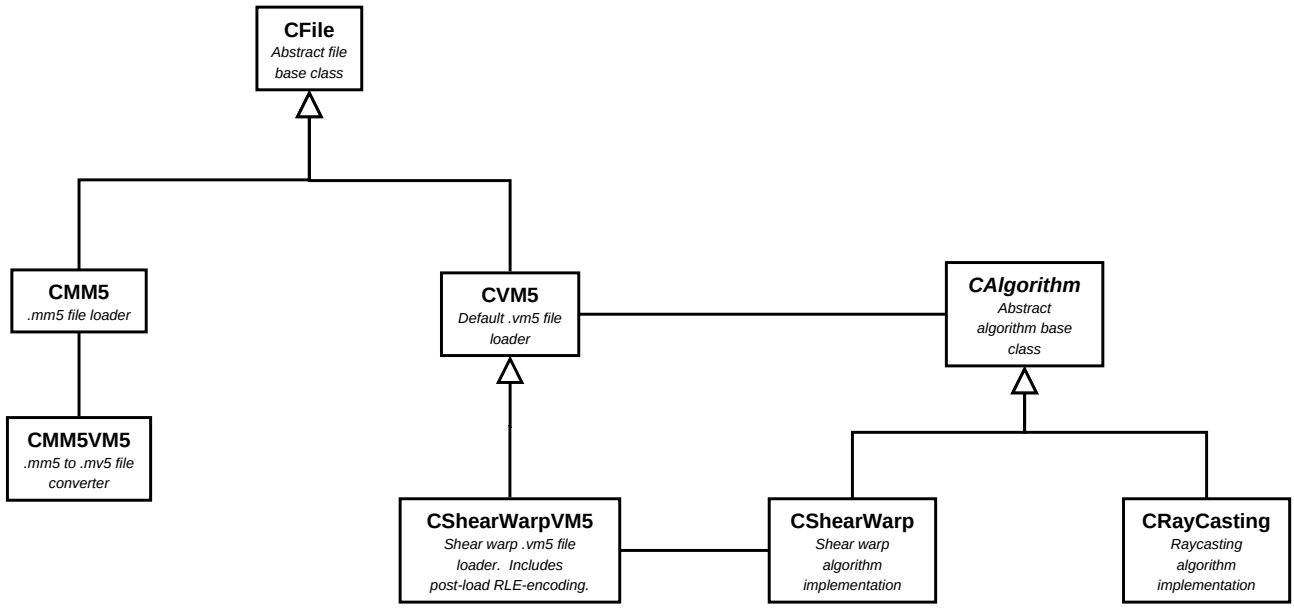


Figure 5. Software Design

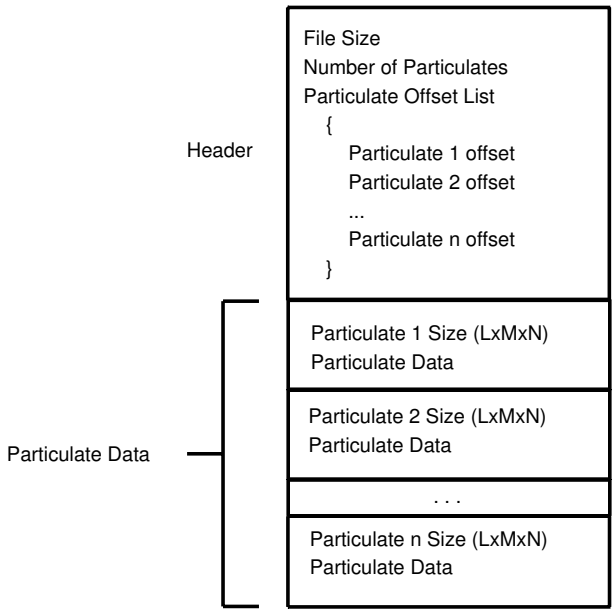


Figure 4. File Structure

CONCLUSIONS

Vesuvius provides atmospheric scientists a chance to interactively evaluate the large amounts of data that are generated from an atmospheric simulation. Instead of gigabytes of raw numbers they can utilize the three-dimensional interaction and visualization capability provided by FLEX and VisBox-P1 or possibly other virtual reality environments.

By allowing them to play back the entire simulation as well as choose particulate opacity and which particulates to enable and disable, atmospheric scientists are given an incredible degree of control regarding what they choose to study and look at. In addition, by rendering the entire simulation in a full 3D virtual reality environment they are allowed an unparalleled view from any angle with amazing detail at the interactions between particulates in a simulation and interactions of the particulates with surrounding terrain.

FUTURE WORK

Optimizing the renderer to achieve higher framerates is the main focus of our future work. By paralellizing the algorithm to fully utilize the many CPUs available to us, the computation of the volume rendering should be done in far less time. Also, by rethinking how data is shared between the different screens in the virtual environment we hope to minimize redundant data in memory. We feel that by re-evaluating how overlapping volumes affect each other we can come up with a new, less computationally expensive process. A texture-based volume rendering algorithm is also being considered for hardware acceleration of small datasets or small particulates in a dataset. In addition to rendering optimizations, optimizations can also be made to the file structure. By reorganizing the custom volume file we hope to be able to fit more into memory and have less hard drive access and seeking, which are notoriously slow.

ACKNOWLEDGEMENTS

This work was partially supported by the US Army PEO-STRI, NAVAIR Contract N61339-04-C-0072

REFERENCES

Boada, I., Navazo, I. and Scopigno, R. 2001. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, *17(3)*, 185–197.

Chen, J., Fang, Y., Loftin, R., Leiss, E., Lin, C. and Su, S. 2001. An immersive virtual environment training system on real-time motion platform. *Proc. of the Computer Aided Design and Computer Graphics*, *2*, 951–954.

Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T. and Nishita, T. 2000. A simple, efficient method for realistic animation of clouds. In *SIGGRAPH '00: Proceed-*

ings of the 27th annual conference on Computer graphics and interactive techniques (pp. 19–28). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Ebert, D. S. and Parent, R. E. 1990. Rendering and animation of gaseous phenomena by combining fast volume and scanline a-buffer techniques. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (pp. 357–366). New York, NY, USA: ACM Press.

Fang, S., Srinivasan, R., Huang, S. and Raghavan, R. 1996. Deformable volume rendering by 3d texture mapping and octree encoding. *IEEE Visualization '96,* (pp. 73–80).

Grimm, S., Bruckner, S., Kanitsar, A. and Gröller, E. 2004. Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In *2004 IEEE Symposium on Volume Visualization and Graphics* (pp. 1–8).

Harris, M. J. and Lastra, A. 2001. Real-time cloud rendering. *Computer Graphics Forum*, *20(3)*, 76–85.

Huang, E. M., Mynatt, E. D., Russell, D. M. and Sue, A. E. 2006. Secrets to success and fatal flaws: The design of large-display groupware. *IEEE Computer Graphics and Applications*, *26(1)*, 37–45.

Knittel, G. 2000. The UltraVis system. In *IEEE Visualization '2000*.

Koller, D., Lindstrom, P., Ribarsky, W., Hodges, L. F., Faust, N. and Turner, G. 1995. Virtual gis: A real-time 3d geographic information system. *vis*, *0*, 94.

Lacroute, P. and Levoy, M. 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics*, *28(Annual Conference Series)*, 451–458.

Levoy, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, *8(3)*, 29–37.

Levoy, M. 1990. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, *9(3)*, 245–261.

Lin, C., Loftin, B., Kakadiaris, I., Chen, D. and Su, S. 2000. Interaction with medical volume data on a projection workbench. In *The Proceedings of the 10th International Conference on Artificial Reality and Telexistence* (pp. 148–152). Taipei, Taiwan.

Loftin, R. B., Pettitt, B. M., Su, S., Chuter, C., McCammon, J. A., Dede, C., Bannon, B. and Ash, K. 1998. Paulingworld: An immersive environment for collaborative exploration of molecular structures and interactions. In *NORDUnet'98, 17th Nordic Internet Conference*.

Meissner, M., Huang, J., Bartz, D., Mueller, K. and Crawfis, R. 2000. A practical evaluation of popular volume rendering algorithms. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization* (pp. 81–90). New York, NY, USA: ACM Press.

Mora, B., Jessel, J. P. and Caubet, R. 2002. A new object-order ray-casting algorithm. In *VIS '02: Proceedings of the conference on Visualization '02* (pp. 203–210). Washington, DC, USA: IEEE Computer Society.

Sherman, W. R. 2007. Freevr.

Sherman, W. R. and Craig, A. B. 2003. *Understanding Virtual Reality*. San Francisco, CA, USA: Morgan Kaufmann Publishers.

Sobierajski, L. M. and Avila, R. S. 1995. A hardware acceleration method for volumetric ray tracing. *vis*, *0*, 27.

Wan, M., Kaufman, A. and Bryson, S. 1999. High performance presence-accelerated ray casting. In *VIS '99: Proceedings of the conference on Visualization '99* (pp. 379–386). Los Alamitos, CA, USA: IEEE Computer Society Press.

AUTHOR BIOGRAPHIES

MICHAEL P. DYE was born in Orange, California, USA. He received his BS in Computer Science from the University of Nevada in 2005, and is currently an MS student in Computer Science and should finish in August 2007. He has been employed in the Center for Advanced Visualization, Computation, and Modeling (CAVCaM) at DRI for the past two years. His research is in graphics, computer games, and immersive scientific visualization. His e-mail address is: `mdye@cse.unr.edu`.

FREDERICK C. HARRIS, JR. was born in San Jose, California, USA. He received his BS in Mathematics and MS in Educational Administration from Bob Jones University in 1986 and 1988, and his MS and PhD in Computer Science from Clemson University in 1991 and 1994. He has been at the University of Nevada, Reno since 1994, and is currently a Professor in the Computer Science and Engineering Department. He also has an affiliated faculty position with Desert Research Institute in Reno, where he is in the Center for Advanced Visualization, Computation and Modeling. His research is in the areas of Parallel and Distributed Computing, as well Graphics and Virtual Reality. His e-mail address is: `fredh@cse.unr.edu` and his Web page can be found at `http://www.cse.unr.edu/~fredh`

PHILIP A. McDONALD was born in Lebanon, Indiana. He received his BS in Forestry from the University of California, Berkeley in 1971 and his MS in Environmental Design and Meteorology from the University of Oklahoma in 1984. He has been a visualization software engineer serving the atmospheric sciences community for more than twenty years. He has been a faculty member in the Center for Advanced Visualization, Computation and Modeling at the Desert Research Institute since 2005 where he is an Assistant Research Visualization Scientist. His research areas include the application of advanced visualization methods to the earth sciences. His email address is: `phil.mcdonald@dri.edu`

WILLIAM R. SHERMAN was born in Madison, Wisconsin, USA. He received his BS in Computer Engineering and MS in Computer Science University of Illinois in 1986 and 1988 respectively. He was a Visualization Scientist for the National Center for Supercomputing Applications (NCSA) for 15 years. He came to the Desert Research Institute in 2004 as the Technical and Acting Director of the Center for Advanced Visualization, Computation,and Modeling (CAVCaM). His research areas include Virtual Reality and Scientific Data Visualization. His email address is: `bill.sherman@dri.edu`

A TAXONOMIC MODEL SUPPORTING HIGH PERFORMANCE SPATIAL-TEMPORAL QUERIES IN SPATIAL DATABASES

Gregory Vert, Rawan Alkhaldi, Sara Nasser, Frederick C. Harris. Jr., Sergiu M. Dascalu
Department of Computer Science and Engineering
University of Nevada, Reno
Reno NV 89509, USA
E-mail: {gvert, alkhaldi, sara, fredh, dascalus}@cse.unr.edu

KEYWORDS: spatial data storage, spatial taxonomies

ABSTRACT

Spatial data has become more important everyday in decision-making and planning processes. As such, it needs to be stored and retrieved in information systems that often require high performance due to the voluminous nature of spatial data. Typically this is not much of a problem unless one considers the effect of spatial extent as a function of time in information retrieval. Taxonomies of spatial objects can be useful in suggesting a storage model that addresses spatio-temporal queries. This research develops such a taxonomy and then proposes how the taxonomy might lend itself to a high performance binary tree model for query and storage of spatial data that considers the relationship of time on the shape of objects in storage. The approach has the potential to retrieve data for certain types of queries much more quickly than a linear search of the same types of spatial objects. Comparative evaluation will be the subject of future work.

INTRODUCTION

Spatial databases have become increasingly more important in everyday life. Although they were first used in government and military operations, they have become a basic commodity for almost every business and home. Today, almost everyone knows how to run a computer. Spatial earth resource data has become increasingly important in everyday decision-making and planning processes, such as resource management and urban planning. As such, there is a need to store, retrieve and manage spatial information in an efficient fashion. Due to the volume of information associated with spatial data, geographic information systems (GIS) need to focused on high performance methods for information management. Some methods to date such as Z curves and Hilbert curves attempt to organize spatial data so that it can be retrieved and stored in a linear fashion that reflects the spatial adjacency of geographic objects. These methods however do not address the temporal aspects of spatial data and the effect of time on geometric shape. Additionally, little work has been done on how to retrieve, store and order spatial information that has geometries that change as a function of time. Taxonomies of spatial data can organize information in a fashion that can be utilized for high performance storage and retrieval of temporally oriented spatial data and may have other applications such as spatial data authentication (Vert et al. 2003).

A taxonomy is typically defined in literature as theories and techniques of naming, describing, organizing, and classifying organisms. For example, the biological taxonomic hierarchy describing life is, from top to bottom: kingdom, phylum (for animals) or division (for plants and fungi), class, order, family, genus, species. Data taxonomies are useful because they can simplify and organize access to data. However, taxonomies have been typically developed in the past for a special purpose or application. The goal of this research has been to develop a method and taxonomy that classifies the spatial data contained in maps based on their type, relationship to other spatial objects, and the time effect on such objects. Implied is that the classification of spatial data should be based on similarities of structure or origin as a function of time. This research proposes the concept of classification of spatial data into taxonomies that can then be utilized for rapid retrieval of spatial information within a spatial extent affected by time.

PREVIOUS WORK

A search of the literature for previous work with spatial taxonomies came up with limited results. The University of California, Santa Barbara has developed geographical data taxonomy for the Alexandria Digital Library (ADL). They called the taxonomy "Feature Type Thesaurus," in which a thesaurus is defined as a "set of terms representing concepts and the relationships among the terms, including hierarchy, equivalence, and associative relationships" (Jensen and Snodgrass, 1994). The library uses the thesaurus to organize the geographic information based on the nature of the place. This taxonomy was not based on spatial data objects that exist in maps; it rather classifies almost all geographical terminology that is used to describe places in nature. They used "MultiTes" software to create their "Feature Type Thesaurus." MultiTes is software that has many tools that make it easy to create and manage thesauri, taxonomies, and other types of controlled vocabularies.

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

The following is an example of how the ADL thesaurus (Jensen and Snodgrass, 1994) can be used:

"Sample thesaurus entry with explanations: *Canals* is a feature type category for places such as the Erie Canal. This category is used instead of any of the following:

- Canal bends
- Canalized streams
- Ditch mouths
- Ditches
- Drainage canals
- Drainage ditches
- … More …

Broader terms: Canals is a sub-type of "hydrographic structures."

The following is a list of other categories related to canals (non-hierarchical relationships):

- Channels
- Locks
- Transportation features
- Tunnels

So, canals can be defined as manmade waterway used by watercraft or for drainage, irrigation, mining, or water power."

Calkins and Obermeyer developed a taxonomy of spatial data use and value that aims at enhancing the general understanding of importance and use of geographic information in decision making (Calkins and Obermeyer, 1991). Onsrud and Rushton have suggested a taxonomy of spatial data sharing based on the characteristics of organizations, data, constraints, and exchange that is intended to differentiate the different activities of spatial data-sharing to build future relationship models (Onsrud and Rushton, 1995).

Jensen and Snodgrass have attempted to address time in the spatial context by creating a temporal taxonomy for events and classified them based on a valid time and a transaction time into fifteen different categories. These categories review generalized temporal relations in order to be able to query a predecessor relation from a successor relation for the purposes of information retrieval (Jensen and Snodgrass, 1994).

Peuquet classified temporal objects (Heywood, et al. 2002), based on the type of event that caused the change to the entity (point, line, or polygon), into four types:

- Continuous - events that occur over a period of time.
- Majorative – events that go on most of the time.
- Sporadic – events that occur some of the time.
- Unique – events that occur only once.

For example, for swimmers who go to the beach, swimming would represent a majorative event during a single day because this event (swimming) occurs most of the time during the day. But, for swimmers to visit restaurants and cafes this event could be considered sporadic because it will occur for limited times during the day for breakfast, lunch, and dinner. The existing approaches to spatial taxonomies seem to be created for special purposes and do not actually classify spatial objects but aspects and events related to spatial objects. What has been needed is a new approach that classifies spatial objects in a robust fashion, hierarchically, and considers the nature of time on such objects. The development of such taxonomy can potentially be applied to rapid, high performance wild card queries of spatial databases by suggesting a tree structure for data organization

APPROACH

In developing of this taxonomic approach, several challenges had to be identified and addressed. The first of these was that of trying to collect and list the spatial data objects used in maps. Firstly, there appeared to be an endless number of spatial object types. Upon examination of the data, these were determined to stem from the fact that there appears to be no naming standards for spatial objects. The same object may exist under a different name in a different map. For example, a small lake in one data file may be referred to as a pond in another file. Most of the spatial data examined in this research was from the USGS and Geography departments at the University of Nevada. Additionally, each taxonomic class needed the concept of a time dimension applied to it. For example there are bodies of water that flow occasionally above ground and occasionally below ground in the deserts of the western United States based on the time of year.

The first step of our research was to build a collection of spatial object identifiers and then look for structural classifiers. Because little previous work had been done in this regard we started by examining objects from the previously mentioned taxonomies (Heywood et al., 2002, Calkins and Obermeyer, 1991) and by examining USGS data for the state of Nevada and ESRI spatial objects (ESRI, 2007). The initial categories defined from this work are: Urban, Rural, Forest, Soil, Costal, and Polar. These categories were determined based on the Alexandria Digital Library Feature Type Thesaurus (Jensen and Snodgrass, 1994). For classification we examined the data found in the ESRI Data & Maps Media kit (ESRI, 2002, ESRI, 2007). After collecting the data for the taxonomy, it was easy to distinctly categorize the data types into the categories developed in previous work (Tables 1-7). In addition, it was realized that those groups, which had been developed for purposes other than taxonomies, would not work correctly for data classifications.

Table 1: Hydrography spatial datasets

Lake	River	Ocean
Sea	Bay	Strait
Fiord	Inlet	Sound
Pond	Stream	Swamp
Estuary	Lagoon	Harbor
Shore	Port	Waterfall
Creek	Cove	Dam/weir
Brook	Ditch	Anchorage
Beach	Island	Ice mass
Geyser	Reef	Hot spring
Reach	Desert water	Oases
Gulch	Gulf	Channel
Reservoir	Marsh	Slough
Cape	Canal	Bayou
Bog	Playa	Desert Lakes

Table 2: Urban spatial datasets

Building	Home	Gauging Stations
Fort	University	Telephone Lines
City	Parcel Data	Transmission Lines
Estate	Power Lines	Sewer Sys.& Water Sys.
House	Castle	Communications & Electricity
College	Voting District	National Monument
County	Major Cities	

Table 3: Transportation spatial datasets

Road	Trail	Railroad
Gate	Bridge	Dead end
Lane	Highway	
Crossing	Interchange	Service facility
Blvd	Avenue	Air force base
Bypass	Street	Interstate
Transit	Station	Launch complex
Junction	Fork	Airport
Freeway	Residential	Dirt
Train yard	Route	Drive
Tunnel	Exit	Way
Jeep Trail	Train tracks	Landing strip

Table 4: Vegetation spatial datasets

Park	Lawn	Garden
Land	Timber	Farm
Reserves	Grove	Field
Bushes	Brushwood	Vineyard
Pasture	Cultivated areas	Ranch
Croplands	Zoo	Forest
Wood	Trees	Meadow
Orchard	Plantation	

Table 5: Rural spatial datasets

County	Septic sys. & Wells	Parcel Data
Village	Transmission Lines	
Barn	Gauging Station	Dairies
Corral	Communications & Electricity	Voting Districts
Estate	Telephone Lines	House
Building	Power Lines	
Hamlets	Town	

Table 6: Soil spatial datasets

Clay	Sand sea	Bedrock outcrop
Sand	Silt	Alluvial fans
Rocks	Desert soil	Sand sheet

Table 7: Elevation spatial datasets

Hill	Valley	Mountain
Summit	Gap	Canyon
Mound	Crest	Coulee
Butte	Volcano	Crevasse
Mine	Ridge	Quarry
Mount	Dale	Pike
Foothill		

Grouping of spatial data into these categories was done by structural or semantic basis. For example, man-made structures found in urban areas became members of the urban spatial dataset. Whereas, the hydrography category of spatial objects contains objects that can have varying geometries and thus are classified by the semantics of being objects that are characterized semantically by water. With the spatial data categorized, we then classified each type of data object based on its geometric stability over a given unit of time. Tables 8 (a) and (b) show this the result of this classification. The abbreviations and symbols used in these tables are as follows:

Legend

I ➔ Innate
RE ➔ Relative Extent
S ➔ Seasonal
C ➔ Continual
Sing. ➔ Singular
Y ➔ Yes
L ➔ Large
V ➔ Vary
Blank ➔ N/A
S ➔ Static
ST ➔ Static-Temporal
TC ➔ Temporal-Continuous
TS ➔ Temporal-Sporadic

Table 8 (a): Temporal classification of spatial data classes from the taxonomy

Class	I	RE	S	C	Sing.	TG
Ocean	Y	L				**S**
Ice Mass		L	Y			**TC**
Sea	Y	V				**ST**
Lake		V	Y			**TC**
River		V	Y			**TC**
Desert Water		V	Y			**TC**
Island	Y					**S**
Shore	Y			Y		**TC**
Port					Y	**TS**
Dam/Weir					Y	**TS**
Sand	Y			Y		**TC**
Silt			Y			**TC**
Clay			Y			**TC**
Rocks	Y					**S**
Land		V			Y	**TS**
Forest	Y	L				**S**
Farm		V			Y	**TS**
Park		V			Y	**TS**
Bushes		V	Y			**TC**
Summit	Y					**S**
Mountain	Y	L				**S**
Hill	Y	V				**S**
Valley	Y	V				**S**
Mine		V			Y	**TS**

Grouping spatial objects by temporal attributes produces the classification shown in Table 9. This classification can then be used to order the objects in a category by their spatial extent and the effect of time on the spatial extent. For example, an ice mass is large and is continuously changing spatial extent over a period of time.

In the classification process used for the taxonomy, temporality was studied based on relative features amongst the classes within the same natural category. By mapping classes from each group onto the spatial-temporal plots, classes could be compared on the basis of spatial extend and time. The spatial-temporal plots are logarithmic representations of the relative size and temporality of spatial objects that can show the relative relationship between different objects.

The spatial-temporal plots use the log scales of area and time as their axis. For example, classes of the static-temporal group had been studied on the spatial-temporal plot shown in Figure 1. The Ocean class has been placed on the top right corner because it is large in size and changes in its extent happen over a very long period of time. Island, mountain, forest, hill, and valley classes have been clustered in the middle of the spatial-temporal plot. The classes of rocks and summit have been placed at the lower left corner. The presented mapping is based on our interpretation of the different classes; other interpretations may produce different outcomes. For simplicity purposes, Figure 1 can be represented in aggregate form as shown in Figure 2.

Table 8 (b): Temporal classification of spatial data classes from the taxonomy

Class	I	RE	S	C	Sing.	TG
County		L			Y	**TS**
City		L			Y	**TS**
University		V			Y	**TS**
Building		V			Y	**TS**
National Monument					Y	**TS**
Voting District		L			Y	**TS**
Parcel Data		L			Y	**TS**
Sewer and Water Systems					Y	**TS**
Communication and Electricity					Y	**TS**
County		L			Y	**TS**
Village		L			Y	**TS**
Town		L			Y	**TS**
Building		V			Y	**TS**
Corral		V			Y	**TS**
Voting District		L			Y	**TS**
Parcel Data		L			Y	**TS**
Septic Systems and Wells					Y	**TS**
Communication and Electricity					Y	**TS**
Road					Y	**TS**
Trail					Y	**TS**
Crossing					Y	**TS**
Railroad					Y	**TS**
Airport		V			Y	**TS**
Bypass					Y	**TS**
Service Facility		V			Y	**TS**

All classes from the developed taxonomic classification with similar temporal classifications have been represented on the spatial-temporal plot as shown in Figures 1, 2, 3, 4, and 5.

Table 9: Temporality classification of the taxonomy

Static	Static-Temporal	Temporal-Continuous	Temporal-Sporadic
Ocean	Sea	Ice mass	Port
Island		Desert water	Land
Rocks		Shore	Farm
Forest		Sand	Park
Summit		Silt	University
Mountain		Clay	Parcel data
Hill		Bushes	Corral
Valley		Lake	Dam/Weir
		River	Mines
			County
			Building
			National Monument
			Voting Districts
			Road
			Crossing
			Railroad
			Bypass
			Service Facility
			Septic sys/ Wells
			City
			Airport
			Village
			Town
			Trail
			Sewer sys/ water sys.
			Communications & Electricity

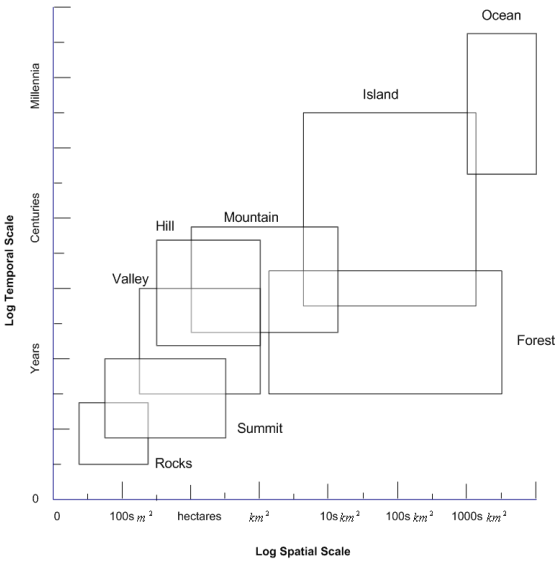


Figure 1: Spatial temporal plot for classes with Static (S) classification

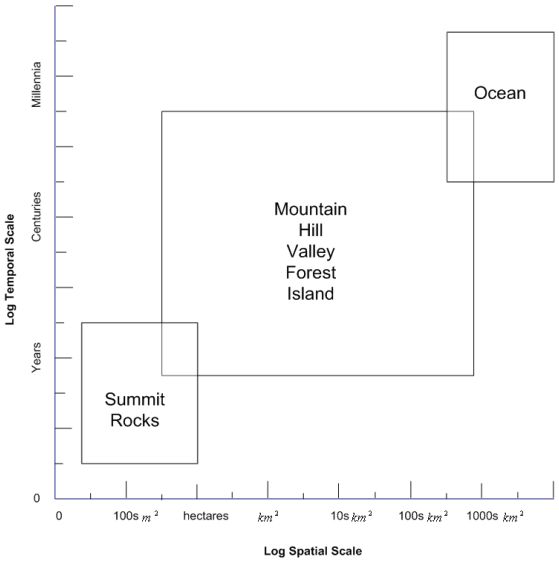


Figure 2: Aggregated representation of spatial temporal plot for static group in Figure 1.

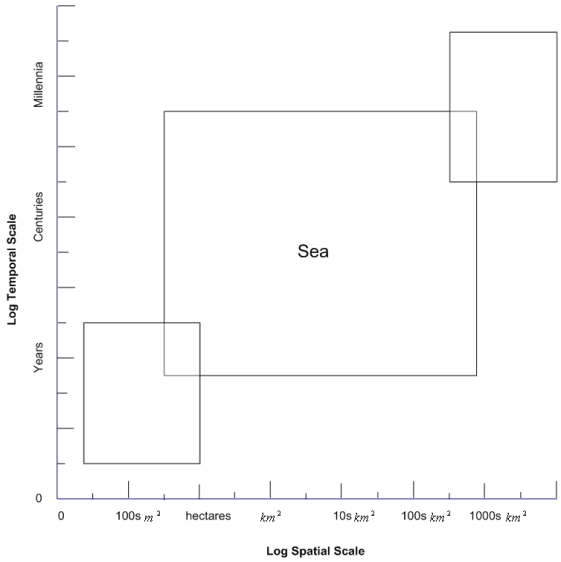


Figure 3: Aggregated spatial temporal plot for static temporal (ST) classification

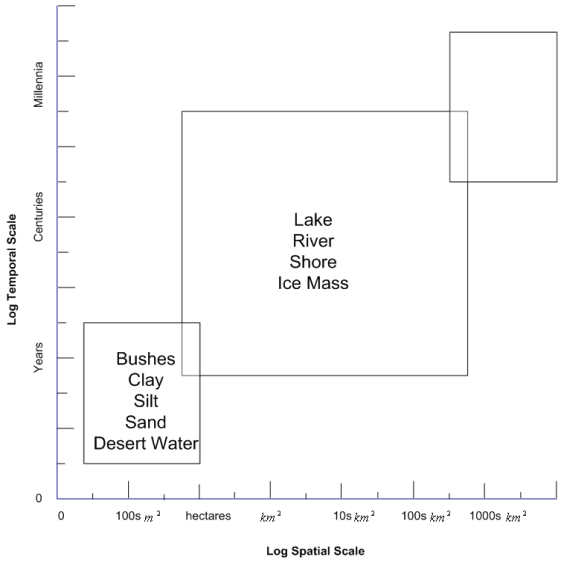


Figure 4: Aggregated spatial temporal plot for temporal continuous (TC) classification.

APPLICATION

The temporal spatial plots developed in Figures 1-5 can be used for the design and creation of binary storage structures that can be used to support high performance range and bound queries of spatial data. To illustrate the application, imagine a query of the form:

*Retrieve * Static objects with spatial extent > 1 hectacre on time scale > n years*

Using Figure 2, this query would traverse a tree structure for Static object and return Mountain, Hill Valley, Forest, Island and Ocean objects. While the subject of future research, the spatial temporal taxonomies seems to suggest a binary tree structure (Figure 6) whose worst search time would be `O(log n)` versus `O(n)` for a linear search of the same data. Such a tree structure might look like the one shown in Figure 6, where objects are organized by spatial extent and trees are characteristic of the temporal classification of objects in the tree.

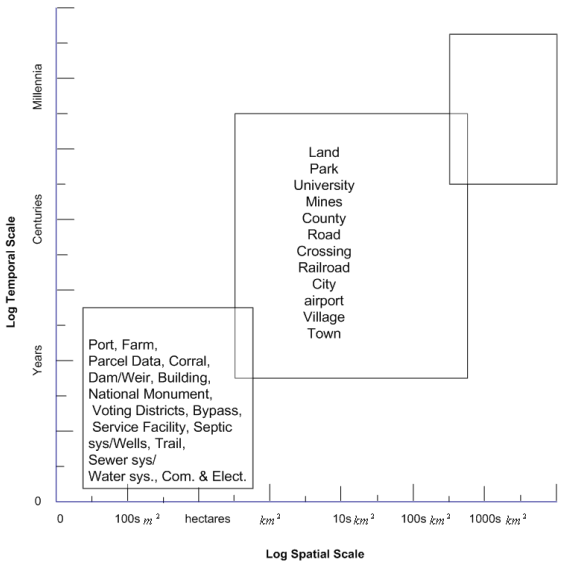


Figure 5: Aggregated spatial temporal plot for temporal sporadic (TS) classification

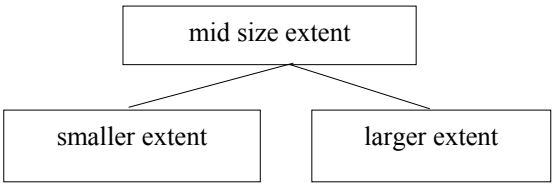


Figure 6: Architectural search tree based on taxonomies

When candidate objects are found in a node in the above search tree, a similar binary search is run to meet the temporal taxonomic criteria of the query as shown in Figure 7.

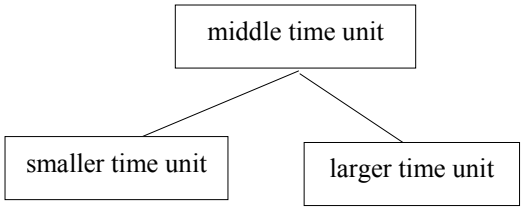


Figure 7: Temporal taxonomic search tree

This can then produce a search time for spatial temporal queries of `O(lg (n + m))` where n is the number of nodes in the spatial tree and m is the number

of elements in the spatial tree. In contrast a linear search would run in `O(n+m)` time. Consequently, taxonomies based on spatial extent as a function of time, appear to be useful in developing data structures and organization that can support high performance queries. Additionally, such an approach incorporates the concept of temporality in a way that previous approaches do not. Evaluation and empirical measurement of exactly how much faster this approach is will be the subject of future research.

FUTURE WORK AND CONCLUSIONS

The spatio-temporal taxonomy developed in this research classifies spatial data objects by their physical extents and by their temporal properties. This classification is then turned into a taxonomy which leads to the organization of the taxonomy into spatio-temporal plots. These plots suggest a way to organize spatial data that supports queries to a database based on time and the physical extent of a spatial object as a function of time. The suggested performance improvement is dramatic versus a linear search and provides support not found in spatial data queries such as Z curves and Hilbert curves. Future work will include more precise techniques for quantification of relationships in the taxonomy. Additional work will be done to empirically measure the performance of this technique versus other storage structures.

REFERENCES

Calkins, H.W. and Obermeyer, N.J., 1991, Taxonomy for surveying the use and value of geographical information. *International Journal of Geographic Information Systems* **5**(3): 341-351.

ESRI, 2007, Introduction to ArcView 3.x., ESRI Virtual Campus, GIS Education and Training on the Web. Retrieved March 19, 2007 from: http://campus.esri.com/

ESRI, 2002, Data & Maps, Media Kit, ESRI ArcGIS. http://www.esri.com

Heywood, I., Cornelius, S. and Carver, S., 2002, *An Introduction to Geographical Information Systems*, 2nd Edition, Prentice Hall.

Jensen, C.S. and Snodgrass, R., 1994, Temporal specialization and generalization. *IEEE Transactions on Knowledge and Data Engineering* **6**(6): 954-974.

Onsrud, H.J. and Rushton, G., 1995, Sharing Geographic Information. Center for Urban Policy Research, New Brunswick, N.J. pp. 510.

Takeyama, M. and Couclelis, H., 1997, Map dynamics: integrating cellular automata and GIS through Geo-Algebra. *International Journal of Geographical Information Science* **11**: 73-91.

Vert, G. Yuan, B. Cole, N., 2003, "A Visual Algebra for Detecting Port Attacks on Computer Systems," Proceedings of the Intl. Conf. on Computer Applications in Industry and Engineering (CAINE-2003), Las Vegas, NV, pp 131-135.

AUTHOR BIOGRAPHIES

GREGORY VERT was born in Fairfield California. He received his BS in Geography with a specialization in GIS from the University of Washington in 1985, his MS in Information Systems Management, Seattle Pacific University, in Seattle, Washington in 1988, and his PhD in Computer Science from the University of Idaho in Moscow Idaho in 2000. He has been an Assistant Professor at the University of Nevada, Reno in the Computer Science and Engineering Department since 2002. His research is in the areas of GIS, Computer Security, Fuzzy System, Database and Bioinformatics. His email address is `gvert@cse.unr.edu`

RAWAN ALKHALDI was born in Jordan. She received her MS in Computer Science from the University of Nevada in 2006. Her research is in the area of Spatial Databases and Spatial Taxonomies. Her email address is `alkhaldi@cse.unr.edu`

SARA NASSER was born in Hyderabad, India. She received her BE in Computer Science and Engineering from Muffakham Jah College of Engineering and Technology, Osmania University, and her MS in Computer Science from the University of Nevada in 2003. She is currently a PhD student in Computer Science and Engineering and is expecting to graduate in 2008. Her research is in the area of Bioinformatics and Fuzzy Systems. Her e-mail address is `sara@cse.unr.edu`

FREDERICK C. HARRIS, JR. was born in San Jose, California, USA. He received his BS in Mathematics and MS in Educational Administration from Bob Jones University in 1986 and 1988, and his MS and PhD from Clemson University in 1991 and 1994. He has been at the University of Nevada, Reno since 1994, and is currently a Professor in the Computer Science and Engineering Department. His research is in the areas of Parallel and Distributed Computing, as well Graphics and Virtual Reality. His e-mail address is: `fredh@cse.unr.edu`

SERGIU M. DASCALU was born in Bucharest, Romania. He received his MS in Automatic Control & Computers from the Polytechnic of Bucharest, and his PhD in Computer Science from the University of Dalhousie, Halifax, Nova Scotia, Canada in 2001. He has been at the University of Nevada since 2002 and is currently an Assistant Professor in the Computer Science and Engineering Department. His research is in the areas of Software Engineering and Human-Computer Interaction. His email address is: `dascalus@cse.unr.edu`

MANAGING DATA AND COMPUTATIONAL COMPLEXITY FOR IMMERSIVE WILDFIRE VISUALIZATION

Michael A. Penick[1,2] Roger V. Hoang[1,2]
Frederick C. Harris Jr.[1,2] Sergiu M. Dascalu[1]
Timothy J. Brown[2] William R. Sherman[2] Philip A. McDonald[2]

[1]University of Nevada, Reno
Department of Computer Science and Engineering
Reno, NV 89557

[2]Desert Research Institute
Center for Advanced Computation and Modeling
Reno, NV 89512

penick@cse.unr.edu

KEYWORDS
Virtual Reality, Wildfire Simulation, Immersive Systems

ABSTRACT

Wildfires are a concern for communities throughout the world. They cause millions of dollars in damage and lead to loss of lives. The development of computational models to predict wildfire behavior is necessary to minimize wildfire damages and casualties. Visualizing the data generated from these computational models has many applications including training, strategic planning, data analysis, and model validation. The complexity of visualizing wildfire brings many challenges, further complicated by large datasets and specialized hardware used to drive immersive systems. This paper present methods for managing the large datasets and computational complexity involved in visualizing large wildfires in immersive environments.

INTRODUCTION

Wildfires are very unpredictable. It is difficult to determine exactly where and when a wildfire will happen next and it is even more difficult to determine how a wildfire will spread with absolute precision. It is the unpredictability of wildfires that make them so dangerous. It is this reason that so much time and money is spent researching wildfire behavior.

There are many advantages to modeling the spread of wildfires. Spread models can be used to develop plans to fight fire, initiate more predictable prescribed burning, and also predict the risk involved if a wildfire occurred in a certain area. Determining how much risk to an area's inhabitants and their property can be used to spend money appropriately and develop a proactive plan for evacuation and fire fighting. Kyle Canyon in Southern Nevada is a good example of a high danger wildfire zone. In the event of a wildfire, it would be very difficult, if not impossible, to evacuate its citizens and would most likely end in a high fatality rate. Many decisions rely on the results of wildfire spread model simulations. It is important that these models, to some degree, accurately predict the spread.

Validating these models is difficult without the wildfire actually happening and comparing the results. Visualization of the wildfire model output in an immersive environment can be used to validate its output against environment factors such as terrain slope, fuel moisture, wind vectors and weather conditions. It can also be used to compare model output against video footage or a visual recreation of the scene from collected data. This is only a single but important reason for visualization of wildfire model output. Visualization of these model outputs can be used to better train firefighters and fire bosses and aid in burning more predictable prescribed fires.

Burning a wildfire for the purpose of training is dangerous and costly. Virtual reality technology makes it possible to recreate wildfire scenarios with more realistic results then previously possible. Recreating wildfire or using model outputs can be used to train fire bosses and firefighters and aid in the development of plans and precautions. With the development of a real-time wildfire simulation it would be possible to run through several virtual scenarios very quickly. This could be used to better determine what measures to take while burning a prescribed fire.

Real-time visualization of this data is a necessary requirement for these applications. Faster than real-time visualization would also bring many advantages. Wildfires often burn over large areas of land even covering tens to hundreds of thousands of acres resulting in the need to visualize of several large datasets. Visualizing terrain and forests of this magnitude is a computationally intensive task while maintaining real-time frame rates. Rendering a fire across this vast landscape also brings many challenges.

The rest of this paper is structured as follows. The next section presents related work on wildfire visualization, virtual reality, possible hardware configurations, and our software environment. This is followed by a section describing our implementation of different parts of the wildfire scenario. The paper finishes with our conclusions and possibilities for future work.

BACKGROUND

Fire and Wildfires

Much work exists for visualizing fire for the purpose of training and analysis. However, little work has been done to visualize fires and wildfires in an immersive medium. A good amount of work as been spent visualizing com-

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

putational models for in-building fires (Bukowski and Sequin, 1997; Govindarajan et al., 1999). (Julien and Shaw, 2003; Tate et al., 1997) use virtual environments and realistic spread models for application in firefighting training scenarios; however, the models and fire visualization are not applicable to outdoor, large-scale fires.

Los Alamos National Laboratory developed a tool for the visualization of wildfire data; however, both their model and visualization ran slower than real-time (Ahrens et al., 1997; McCormick and Ahrens, 1998). This work states the applicability of visualization of wildfire to training, but only describes the graphical elements necessary. Similar work using GIS based reconstruction of forest landscape scenarios has been done with non-immersively visualized wildfires (Yu et al., 2004). The authors chose to implement their own elliptical wildfire spread model based on the Huygen's principle of wave propagation and compute localized fire behavior using the Rothermel model. They achieved real-time frame rates on a desktop system using a custom forest level-of-detail system and wildfire spread model. However, the paper did not discuss the validity of their wildfire spread model and did not address the application or complexities of their application in an virtual environment.

Farsite

FARSITE is a well-established fire behavior and growth simulator developed by the USDA Forest Service. It is used by fire analysts from the US Department of the Interior, National Park Service, US Department of the Interior Bureau of Indian Affairs. (Community, 2007) Its importance and widespread use among fire professionals was a critical factor for choosing to visualize its simulation output. FARSITE incorporates models of surface fire, crown fire, point-source fire acceleration, spotting, and fuel moisture to calculate the spread of fire of a landscape (Finney, 1998). These various models are used to propagate vectors of fire parameter polygons. Intervals of these expanding polygons are interpolated to generate raster data that describe fire behavior. The raster data outputs are stored in ESRI ascii files, of which six of the outputs are of importance to visually reconstructing the wildfire scenario. Several of the inputs required to run a FARSITE simulation are also crucial to visualizing the wildfire scenario. This includes the digital elevation model data used to render the terrain and the fuel load data used to place vegetation.

Virtual Reality

Virtual Reality (VR) as it relates to this project is the use of hardware and software technologies used to allow user to view and interact with computer-simulated environments. The goal of VR is to mentally immerse a user within these environments through different types of sensory feedback. Sight (visual), a type of sensory feedback important to this, but in a broader sense feedback also includes aural (sound), touch (haptic), and smell (olfactory). The broad definition of VR includes the visualization of a 3D world on a desktop computer because this can and often creates a sense of immersion. However, the specific definition used in this project involves the visualization on stereoscopic displays

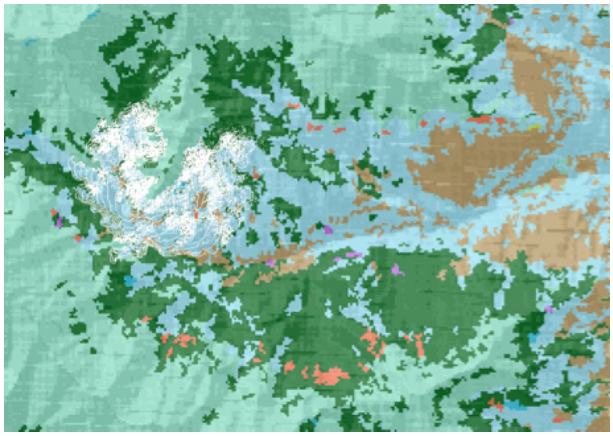


Figure 1. FARSITE Data Visualization

and interaction captured by tracking systems because these technologies offer many advantages over visualization on a desktop system. The source of these advantages is through the support of depth cues not achievable on non immersive systems.

Depth cues are the important information a viewer uses to discern distances between objects in a scene. The more depth cues a system can support the more potential for immersion is possible. Monoscopic, stereoscopic, and motion depth cues are important to VR systems. Monoscopic depth cues are achievable on both immersive and desktops systems. Monoscopic depth cues can be seen in a single static image of a scene. These are the depth cues, which can be drawn from image features such as size, shading, and occlusions. Stereoscopic image depth cues are the differences determined between the images obtained by each eye (left and right images). Motion depth cues can be seen when a viewer changes the relative position between them and an object. The viewer can gauge the distance of an object based on how fast an object passes by when they change perspective (closer objects appear to move faster than farther ones). A 3D desktop environment is only able to provide monoscopic depth cues, but an immersive system with stereoscopic displays and head tracking can simulate all depth cues discussed. The addition of these depth cues allows for an experience closer to reality, making such an environment suitable for training applications.

Immersive Hardware Systems

Multi-screen display systems require specialized hardware unlike HMDs, which can use hardware similar to desktop PCs. Multi-screen display systems require multiple graphics pipes to keep real-time frame rates. The goal is to keep performance independent of the number of screens a system contains. A 6-wall system should have comparable performance with a 4-wall system if driven by similar hardware. Two configurations exist for achieving this goal: The first is a shared memory system with multiple graphics pipes and the second, more recent, is provided by cluster-based systems. Shared memory systems support a single large memory image across all processors. On these systems a process is used to render each screen independently and one to many

processes are used to update the simulation. Rendering each screen in parallel offers performance, which is independent of the number of screens in the system. Performance is further increased by allowing the simulation and rendering to run in parallel. The idea is to run the update and rendering computation in parallel as much as possible, but because they are accessing the same data, the update writing and the rendering reading it must be locked. Cluster-based solutions run the simulation and rendering code on a node for each screen in parallel and a head node keeps the simulations in sync. It is also possible on these systems to run the rendering and simulation code in parallel for increased performance if multiple processors are available. The shared memory solution has the disadvantage of requiring expensive specialized hardware to support multiple graphics cards. High performance, available commodity hardware can be used in cluster-based systems with a fast interconnect.

VR Toolkits and Scene Graphs

VR systems contain specialized input and tracking hardware, but also specialized screen and computational hardware configurations making writing software for these systems a monumental task. The hardware configurations for these systems can vary drastically between different organizations requiring software to be changed for each of these systems. VR toolkits attempt to abstract these differences so that an application can be written once and run on all of these systems. The largest difference between VR toolkits consists of the type of input hardware and computational configurations they support. William R. Sherman's FreeVR supports a variety of input and tracking systems such as common desktop game pads all the way to high end tracking system such as InterSense's IS-900, but only supports shared memory systems. VRUI and VR Juggler supports similar tracking hardware, but also support cluster-based systems.

Scene graphs uses many optimizations to speed up rendering. Many of these offer different types of culling which quickly determine the visibility of geometry and reduce the amount of triangles need to be rendered. Frustum culling removes geometry outside of the viewers point of view. Occlusion culling removes geometry which is not visible because it is behind other geometry in the scene. Small feature culling removes geometry which are smaller than a particular amount of screen space in pixels. Scene graphs also reduce the amount of expensive state changes such as changing shaders, textures, and other rendering states. They also often implement lazy state updating, which only updates states that are not already set. Optimization traversal usually run during a single time after initialization can optimize a the scene graph data structure and the geometry contained within. These optimizations can include organizing the scene graph into an octtree for optimized frustum culling or organizing triangle-based geometry into optimized triangle strips.

Although there are many scene graphs, there are few which are open source and suited to the development of virtual reality applications. Support for multi-pipe rendering is the fundamental feature necessary for rendering on our shared memory system. This support includes management of OpenGL objects (Texture object, Vertex buffer object, etc.) and, less importantly, a data protection mechanism. Data protection is often very specific to a problem to achieve high performance. Both OpenSG and OpenSceneGraph are two scene graphs that meet this criteria. OpenSG implements a system, which allow the scene graph to be transparently shared across multiple machines in a cluster or server processes on a single machine (details about the implementation of this system can be found in these papers (Reiners et al., 2002)). OpenSceneGraph does not have the ability to shared the scene graph data structure, but must be protected using an external locking mechanism or the rendering and simulation update must be done sequentially.

Problem: Visualization of Large Wildfires

Frequently, wildfire can cover large landscapes many times beyond the current computational and memory capacity of current visualization hardware. Each graphical element such as terrain, vegetation, and fire is associate with a large dataset either describing the landscape or driving the simulation. Each one of these element has its own challenges and performance bottlenecks. The digital elevation model data and satellite image describe the terrain. The fuel load data is used to construct the vegetation environment. The wildfire is spread according to FARSITE outputs. Our goal is to achieve real-time visualization of wildfire that cover vast landscape each element much be managed to maintain visual fidelity and run in real-time.

PROPOSED SOLUTION

FreeVR and OpenSceneGraph

We built our application on the open-source FreeVR and OpenSceneGraph (OSG) libraries. The FreeVR virtual reality integration library is a cross-display VR library with built-in interfaces for many input and output devices. It allows programmers to develop on a standard desktop machine, with inputs and display windows that simulate a projection or headbased immersive system. The application can then run just about any type of VR system. The OpenSceneGraph library is used to help with world rendering. OSG allows 3D objects to be hierarchically organized within the environment, and also provides a system that optimizes the rendering through the use of various culling and sorting techniques.

A considerable amount of our effort thus far has been in writing the software interface between FreeVR and OSG. FreeVR works naturally well with OpenGL and other lower level graphical rendering libraries. However, when interfacing a VR integration library with a higher level rendering API there are many issues that need to be addressed, in particular 1) dealing with the perspective matrices, 2) shared memory allocation, 3) multiprocessing, and 4) windowing and input device interfacing. A software interface between FreeVR and the SGI Performer scene-graph library already existed, so we felt confident that the similar OSG library would not be too difficult. The Performer library was

avoided due to its closed-nature, and expected lack of future support.

While the OSG scene-graph system is somewhat based on the efforts of the Performer library, there are two major differences between the OSG implementation and Performer: 1) OSG does not double-buffer the scene-graph, requiring the update traversal to avoid making changes to the scene-graph while a cull traversal is in progress, and 2) because many people contribute new node types to the open-source OSG, there is no strict enforcement of the rule preventing scene-graph modifications taking place outside the update traversal. Neither of these issues is typically a concern for desktop applications running on a single CPU system, but for multi-screen immersive systems, they are problematic. To address these implementation issues, we must specifically avoid the modification of the scene-graph when the multiplerenderings are taking place. FreeVR provides a semaphore-based locking/barrier system that we used to exclude writes to the scenegraph data during culling. Furthermore, when we discovered that some of the node-types (e.g., the particle system node) used the culling traversal to make additional modifications to the scenegraph we had to specifically insert extra locking code into those modules. The end result is a system that works satisfactorily, but the addition of each barrier results in lower frame rendering rates.

Initial Abstractions

To achieve high performance the amount display and simulation computation that should be run in parallel should be maximized. Abstraction of dynamic graphical elements into several steps is necessary to achieve this goal. The processing of each element is broken into three stages: 1) rendering, the code which displays the element 2) updating the simulation code 3) synchronizing and maintaining congruency between the first steps. The first two steps run in parallel working on their own copy of the data and the last step synchronizes the two copies. This uses the assumptions that the simulation code in step 2 is non-trivial and would require more time than syncing the data in step 3. OpenSG provides a similar more generic mechanism; however, we believe that a generic solution is not always possible and a higher framerate is better with some specialization.

Scalable Rendering

Each element of the scene strives to minimize its impact on the performance of the visualization. The goal was to section off as much work to the graphical processing unit as possible and leave the CPU time available for visualization to other elements and simulation code. The following systems manage and minimize their uses of system resource using level-of-detail algorithms specific to their data and visualization domain. The management of resources also allows the system to view landscapes that are larger than the available amount of system resources.

Terrain

In the first implementation of the terrain we chose to implement the Geometrical Mipmapping (de Boer, 2000) algorithm. This enabled the system to visualize terrain datasets much larger than brute-force methods; however, it required that the entire dataset be loaded into memory. This algorithm would result in a complex memory management system. This method although reducing CPU usage over previous methods still could be improved. Ulrich's Chunked Lod algorithm (Ulrich, 2002) offered several advantages including higher triangle throughput, low CPU usage, and implicit memory management. This method, unlike Geometrical Mipmapping, requires offline tessellation of the elevation data. Figure 2 illustrates the tessellation of this approach.

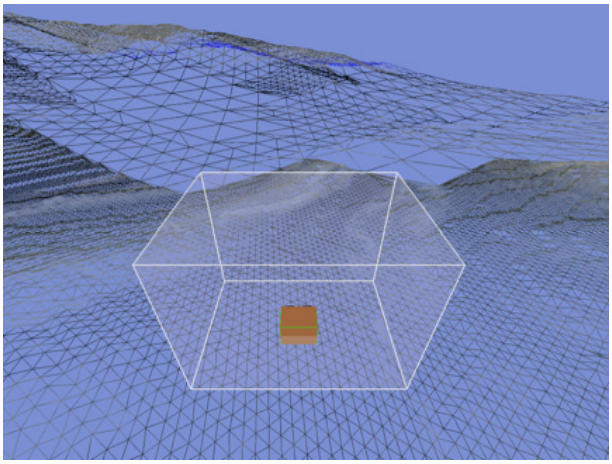


Figure 2. Surface Tessellation

Vegetation

Similarly, vegetation is placed using an offline utility. This saves the cost of having to place trees at runtime and this saving allow the trees to be paged in from a file. The vegetation utility uses the fuel load data input from FARSITE and an expert's knowledge of the location to determine what types of vegetation are native to a location and their positions. Pixel error is used to determine when trees should not be displayed anymore. In general this is when the size of the rendered vegetation is smaller than a few pixels of screen space. Currently, trees are grouped into localized batches to be sent to the video card for rendering. This increases the amount of vegetation able to be rendered, but has the limit of only being able to process vegetation in groups. Processing vegetation individually allows the system to changed the appearance of that vegetation as it burned, but results in slow rendering speeds. A compromise is to use instancing (Corporation, 2004) to draw and process vegetation nearer to the viewer and process distance trees using groups.

Wildfire

Visualization of the wildfire is driven directly from the FARSITE simulation data using particle based fire and smoke. Fire and other natural phenomenon have been successfully rendered using high numbers of particles. Applications using high numbers of particles are not able to render in real-time the use of sprites can significantly reduce the number of particles with good visual results(Reeves, 1983; Feldman et al., 2003). Sprite-based particle systems can accurately and realistically represent fire with very few particles. This is because localized behavior using an accurate offline simu-

Figure 3. Fire and Smoke

Figure 4. Fire Visualization in the Cave

lation and other factor such as color, position, emissiveness, and animation frame can be controlled using a real-time fire model (Wei et al., 2002; Tamas Umenhoffer, 2006; Nguyen, 2004). The visual complexity of particle systems can be reduced progressively as the distance increases from the user; however, the physical properties of all particles must be calculated. Reduction of rendered particles is often the bottleneck because the amount of sprites necessary to render a wildfire quickly reaches the maximum fillrate capacity of the GPU.

RESULTS AND CONCLUSIONS

We have implemented the fire simulation system using the solution decisions described in the previous section. Initial responses from fire agencies have been positive. A screen capture of a sample run showing trees, fire, and smoke is shown in Figure 3. A larger picture of the simulation running in our four-sided Fakespace FLEX system is shown in Figure 4.

Visualization of computational wildfire models has many applications. Wildfire can spread over large amounts of area producing equally large datasets. We have introduced methods that can manage and visualize these datasets interactively. Abstraction of the rendering, simulation computation results in modular code that allows for the integration and optimization of different implementations for rendering elements of a scene. They also take advantage of different hardware configurations used to drive virtual reality systems.

FUTURE WORK

Increasing the accuracy and realism of visualizing wildfire scenarios is a primary focus. This focus will specifically involve the use of additional outputs from FARSITE including flame length, rate of spread and crown fire data. The aim of this endeavor is to improve the accuracy of the visualization for the application of data analysis and model validation. The realistic visual appearance of the fire and smoke is a top priority for presenting wildfire scenarios.

Very little work has been spent on scaleable rendering of realistic real-time fire and smoke with application to the large scale required for wildfires. Current work only considers single or small scale fires over small objects. The complexity of realistically visualizing a single fire can quickly use the entirety of a computer system's resources. We will look to create a specialized LOD system that will further optimize our fire rendering system to support larger wildfires.

The software has been left extensible enough for use with other wildfire simulation models. with the long-term goal of using a real-time model. A real-time wildfire model has many applications including training and predictable prescribed burning. This will also necessitate the inclusion of an enhanced user interface and formal analysis of usability.

ACKNOWLEDGEMENTS

This work was partially supported by the US Army PEO-STRI, NAVAIR Contract N61339-04-C-0072

REFERENCES

Ahrens, J., McCormick, P., Bossert, J., Reisner, J. and Winterkamp, J. 1997. Wildfire visualization (case study). In *VIS '97: Proceedings of the 8th conference on Visualization '97* (pp. 451–ff.). Los Alamitos, CA, USA: IEEE Computer Society Press.

de Boer, W. H. 2000. Fast terrain rendering using geometrical mipmapping.

Bukowski, R. and Sequin, C. 1997. Interactive simulation of fire in virtual building environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (pp. 35–44). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Community, W. F. 2007. FARSITE.

Corporation, N. 2004. GLSL pseudo-instancing. Technical report, Nvidia.

Feldman, B. E., O'Brien, J. F. and Arikan, O. 2003. Animating suspended particle explosions. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (pp. 708–715). New York, NY, USA: ACM Press.

Finney, M. A. 1998. Farsite: Fire area simulator-model development and evaluation. In *Res. Pap. RMRS-RP-4*. U.S. Department of Agriculture, Forest Service.

Govindarajan, J., Ward, M. and Barnett, J. 1999. Visualizing simulated room fires (case study). In *VIS '99: Proceedings of the conference on Visualization '99* (pp. 475–

478). Los Alamitos, CA, USA: IEEE Computer Society Press.

Julien, T. U. S. and Shaw, C. D. 2003. Firefighter command training virtual environment. In *TAPIA '03: Proceedings of the 2003 conference on Diversity in computing* (pp. 30–33). New York, NY, USA: ACM Press.

McCormick, P. S. and Ahrens, J. P. 1998. Visualization of wildfire simulations. *IEEE Comput. Graph. Appl.*, *18(2)*, 17–19.

Nguyen, H. 2004. *GPU Gems* (1st Ed.). Addison-Wesley.

Reeves, W. T. 1983. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, *2(2)*, 91–108.

Reiners, D., Vo, G. and Behr, J. 2002. Opensg: Basic concepts.

Tamas Umenhoffer, L. S.-K. G. S. 2006. Spherical billboards and their application to rendering explosions. In *Proceedings of the 2006 Conference on Graphics Interface* (pp. 57–64). ACM Press.

Tate, D. L., Sibert, L. and King, T. 1997. Virtual reality: Using virtual environments to train firefighters. *IEEE Computer Graphics and Applications*, *17(6)*, 23–29.

Ulrich, T. 2002. Chunked lod: Rendering massive terrains using chunked level of detail control. In *SIGGRAPH '02: Proceedings of the 24th annual conference on Computer graphics and interactive techniques Notes*. ACM Press.

Wei, X., Li, W., Mueller, K. and Kaufman, A. 2002. Simulating fire with texture splats. In *VIS '02: Proceedings of the conference on Visualization '02* (pp. 227–235). Washington, DC, USA: IEEE Computer Society.

Yu, Q., Chen, C., Pan, Z. and Li, J. 2004. A gis-based forest visual simulation system. In *ICIG '04: Proceedings of the Third International Conference on Image and Graphics (ICIG'04)* (pp. 410–413). Washington, DC, USA: IEEE Computer Society.

AUTHOR BIOGRAPHIES

MICHAEL A. PENICK was born in Denver, Colorado, USA. He received his BS in Computer Science from the University of Nevada in 2005 and is currently an MS student in Computer Science and is expected to finish his program of study in May 2007. He has been employed in the Center for Advanced Visualization, Computation, and Modeling (CAVCAM) at DRI for the past two years. His research is in areas of graphics, virtual reality and wildfire visualization. His e-mail address is: `penick@cse.unr.edu`.

ROGER V. HOANG was born in Carson City, Nevada, USA. He received his BS in Computer Science from the University California at Los Angeles in 2006, and is currently an MS student in Computer Science. He has been employed in the Center for Advanced Visualization, Computation, and Modeling (CAVCAM) at DRI for the past year. His research is in areas of graphics, virtual reality and wildfire visualization. His e-mail address is: `hoangr@cse.unr.edu`.

FREDERICK C. HARRIS, JR. was born in San Jose, California, USA. He received his BS in Mathematics and MS in Educational Administration from Bob Jones University in 1986 and 1988, and his MS and PhD from Clemson University in 1991 and 1994. He has been at the University of Nevada, Reno since 1994, and is currently a Professor in the Computer Science and Engineering Department. He also has an affiliated faculty position with Desert Research Institute in Reno, where he is in the Center for Advanced Visualization, Computation and Modeling. His research is in the areas of Parallel and Distributed Computing, as well Graphics and Virtual Reality. His e-mail address is: `fredh@cse.unr.edu` and his web page can be found at `http://www.cse.unr.edu/~fredh`

SERGIU M. DASCALU was born in Bucharest, Romania. He received his MS in Automatic Control & Computers from the Polytechnic of Bucharest, and his PhD in Computer Science from the University of Dalhousie, Halifax, Nova Scotia, Canada in 2001. He has been at the University of Nevada since 2002 and is currently an Assistant Professor in the Computer Science and Engineering Department. His research is in the areas of software engineering and human-computer interaction. His email address is: dascalus@cse.unr.edu

TIMOTHY J. BROWN was born in Springfield, Illinois, USA. He received his BA in Astronomy-Physics from the University of Illinois in 1982, and his MS and PhD in Geography from the University of Colorado in 1988 and 1995 respectively. He has been at the Desert Research Institute since 1995 and is currently an Associate Research Professor. He is currently the Director of the Climate, Ecosystems, and Fire Applications (CEFA). His research areas include climatology and meteorology with an emphasis in wild land fire-climate and fire-weather relationships. His email address is: `Tim.Brown@dri.edu`

PHILIP A. McDONALD was born in Lebanon, Indiana. He received his BS in Forestry from the University of California, Berkeley in 1971 and his MS in Environmental Design and Meteorology from the University of Oklahoma in 1984. He has been a visualization software engineer serving the atmospheric sciences community for more than twenty years. He has been a faculty member in the Center for Advanced Visualization, Computation and Modeling at the Desert Research Institute since 2005 where he is an Assistant Research Visualization Scientist. His research areas include the application of advanced visualization methods to the earth sciences. His email address is: `phil.mcdonald@dri.edu`

WILLIAM R. SHERMAN was born in Madison, Wisconsin, USA. He received his BS in Computer Engineering and MS in Computer Science University of Illinois in 1986 and 1988 respectively. He was a Visualization Scientist for the National Center for Supercomputing Applications (NCSA) for 15 years. He came to the Desert Research Institute in 2004 as the Technical and Acting Director of the Center for Advanced Visualization, Computation,and Modeling (CAVCaM). His research areas include virtual reality and scientific data visualization. His email address is: `bill.sherman@dri.edu`

AUTHOR INDEX