

Prith Banerjee Viktor K. Prasanna
Bhabani P. Sinha (Eds.)

High Performance Computing – HiPC'99

6th International Conference
Calcutta, India, December 1999
Proceedings



Springer

Lecture Notes in Computer Science 1745
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Singapore

Tokyo

Prith Banerjee Viktor K. Prasanna
Bhabani P. Sinha (Eds.)

High Performance Computing – HiPC'99

6th International Conference
Calcutta, India, December 17-20, 1999
Proceedings



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Prith Banerjee
Northwestern University, Electrical and Computer Engineering
2145 Sheridan Road, Evanston, IL 60208-3118, USA
E-mail: banerjee@ece.nwu.edu

Viktor K. Prasanna
University of Southern California
Department of EE-Systems, Computer Engineering Division
3740 McClintok Ave, EEB 200C, Los Angeles, CA 90089-2562, USA
E-mail: prasanna@ganges.usc.edu

Bhabani P. Sinha
Indian Statistical Institute
Advanced Computing and Microelectronics Unit
Computer and Communication Sciences Division
203, B.T. Road, Calcutta 700 035, India
E-mail: bhabani@isical.ac.in

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

High performance computing : 6th international conference, Calcutta, India, December 17 - 20, 1999 ; proceedings / HiPC '99. Prith Banerjee ... (ed.) - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1745)
ISBN 3-540-66907-8

CR Subject Classification (1998): C.1-4, D.1-4, F.1-2, G.1-2

ISSN 0302-9743

ISBN 3-540-66907-8 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1999
Printed in Germany

Typesetting: Camera-ready by author
SPIN 10750013 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

Preface

These are the proceedings of the Sixth International Conference on High Performance Computing (HiPC'99) held December 17-20 in Calcutta, India. The meeting serves as a forum for presenting current work by researchers from around the world as well as highlighting activities in Asia in the high performance computing area.

The meeting emphasizes both the design and the analysis of high performance computing systems and their scientific, engineering, and commercial applications.

Topics covered in the meeting series include:

Parallel Algorithms	Scientific Computation
Parallel Architectures	Visualization
Parallel Languages & Compilers	Network and Cluster Based Computing
Distributed Systems	Signal & Image Processing Systems
Programming Environments	Supercomputing Applications
Memory Systems	Internet and WWW-based Computing
Multimedia and High Speed Networks	Scalable Servers

We would like to thank Alfred Hofmann and Ruth Abraham of Springer-Verlag for their excellent support in bringing out the proceedings. The detailed messages from the steering committee chair, general co-chair and program chair pay tribute to numerous volunteers who helped us in organizing the meeting.

October 1999

Viktor K. Prasanna
Bhabani Sinha
Prithviraj Banerjee

Message from the Steering Chair

It is my pleasure to welcome you to the Sixth International Conference on High Performance Computing. I hope you enjoy the meeting, the rich cultural heritage of Calcutta, as well as the mother Ganges, “the river of life”.

This meeting has grown from a small workshop held five years ago which addressed parallel processing. Over the years, the quality of submissions has improved and the topics of interest have been expanded in the general area of high performance computing. The increased participation and the continued improvement in quality are primarily due to the excellent response from researchers from around the world, enthusiastic volunteer effort, and support from Indian-based IT industries and academia.

It was a pleasure to work with Bhabani Sinha, our general co-chair. He provided me with invaluable input regarding meeting planning and execution and coordinated the local activities of the volunteers in handling numerous logistics.

Prith Banerjee, our program chair, put together a balanced high-quality technical program. Over the past year, I have interacted with him in resolving many meeting-related issues. Vipin Kumar invited the keynote speakers and coordinated the keynotes. Sartaj Sahni handled the poster/presentation session. Nalini Venkatasubramanian interfaced with the authors and Springer-Verlag in bringing out these proceedings. Manav Misra put together the tutorials. I would like to thank all of them for their volunteer efforts.

I am indebted to Sriram Vajapeyam for organizing a timely panel entitled: *Whither Indian Computer Science R&D?*. In addition, he provided thoughtful feedback in resolving many meeting-related issues. R. Govindarajan coordinated the industrial track and exhibits as well as interfacing with potential sponsors.

Over the past year, I have interacted with several volunteers who shared their thoughts with me to define the meeting focus and emphasize quality in building a reputed international meeting. I would like to thank Sriram Vajapeyam, R. Govindarajan, C.P. Ravikumar, D. N. Jayasimha, A.K.P. Nambiar, Sajal Das, and K. R. Venupogal. Their advice, counsel, and enthusiasm have been a major driving force behind this meeting.

Major financial support for the meeting was provided by India-based IT industries. I would like to thank Alok Aggarwal (IBM SRC, India), Suresh Babu (Compaq), Kushal Banerjee (Cisco), S. Dinakar (Novell), Anoop Gupta (Microsoft Research), K. S. Ramanujam (SUN Microsystems India), Karthik Ramarao (HP India), Uday Shukla (IBM India), for their support and participation. I am indebted to Narayana Murthy of Infosys for his continued support.

VIII Message from the Steering Chair

Finally, my special thanks go to my student Kiran Bondalapati for his assistance with meeting publicity and to my assistant Henryk Chrostek for his efforts in handling meeting-related activities throughout the past year.

Viktor K. Prasanna
University of Southern California
October 1999



Message from the General Co-chair

It is my great pleasure to extend a cordial welcome to all of you to this Sixth International Conference on High Performance Computing (HiPC99). This is the first time that the conference on High Performance Computing is being held in Calcutta, the heart of the culture and the economy of Eastern India. I am sure, as in the past five years, this meeting will provide a forum for fruitful interaction among all the participants from both academia and industry. I also hope that you will enjoy your stay in Calcutta, the city of joy.

I express my sincerest thanks to all the keynote speakers who have kindly agreed to attend this conference and deliver keynote lectures on different facets of High Performance Computing.

I am thankful to Prith Banerjee for two reasons: first, for providing his continuous support in holding this conference and second, for performing an excellent job as the program chair of the conference. I would also like to thank Sartaj Sahni for his moral support in holding this meeting and his great service as the poster/presentation chair of the conference.

I am thankful to S. B. Rao, Director of the Indian Statistical Institute, for co-sponsoring this conference providing both financial and infra-structural support. My sincerest thanks also go to all other sponsors including Cisco Systems, Cognizant Technology Solutions, Compaq Computer India Pvt. Ltd., Computer Maintenance Corporation of India (CMC) Ltd., Hewlett-Packard India Ltd., Millenium Information Systems Pvt. Ltd., NIIT Ltd., Novell Software Development (India) Pvt. Ltd., PriceWaterhouseCoopers Ltd., Satyam Computer Services Ltd. and Tata Consultancy Services, but for whom the conference could not have been organized on this scale.

I am grateful to all the members of the local arrangements committee of the conference consisting of Mihir K. Chakraborti (chair), Nabanita Das (convener), Bhargab B. Bhattacharya, Jayasree Dattagupta, Malay K. Kundu, Nikhil R. Pal, Aditya Bagchi, Subhomay Maitra, Rana Dattagupta, Debasish Saha, Rajarshi Chaudhuri, Biswanath Bhattacharya, Tirthankar Banerjee, Subhashis Mitra, Amitava Mukherjee, and Somprakash Bandyopadhyay, who have provided their support in organizing this conference. My special thanks go to Mihir K. Chakraborti for taking active interest in arranging sponsorship for the conference. I am also thankful to Srabani Sengupta and Krishnendu Mukhopadhyaya for extending their help in different aspects related to the conference.

I would like to acknowledge the assistance from A. K. P. Nambiar, finance co-chair of the conference.

I am thankful to R. Govindarajan and Sriram Vajapeyam for their kind co-operation in arranging sponsorship for the conference.

Last but not the least, I would like to thank Viktor Prasanna for his continuous support and advice on all sorts of details in various phases throughout the last year and a half, in order to organize another successful conference on High Performance Computing in India. It was a real pleasure to work with him.

Bhabani P. Sinha
Indian Statistical Institute
General Co-Chair



Message from the Program Chair

I am pleased to introduce the proceedings of the 6th International Conference on High-Performance Computing to be held in Calcutta, India from December 17-20, 1999. The technical program consists of ten sessions, organized as two parallel tracks. We also have six keynote speeches, one panel, two industrial sessions, a poster session, and six tutorials.

The technical program was put together by a distinguished program committee consisting of 49 members and six vice-program chairs. We received 112 papers for the contributed sessions. Each paper was reviewed by three members of the program committee and one program vice-chair. At the program committee meeting, which was attended by the program chairs and program vice-chairs, 60 papers were accepted for presentation at the conference, of which 20 were regular papers (acceptance rate of 18%) and 40 were short papers (acceptance rate of 35%). The papers that will be presented at the conference are authored by researchers from 12 different countries; this is an indication of the true international flavor of the conference.

The technical program consists of three special sessions, two on *Mobile Computing* which were arranged by Sajal Das, and one on *Cluster Computing* which was arranged by Anand Sivasubramaniam and Govindarajan Ramaswamy. While these special session organizers solicited papers for the conference, all papers were subject to the same review process described above.

The program consists of six exciting keynote speeches arranged by the keynote chair, Vipin Kumar. The keynote speeches will be presented by Jack Dongarra on *High-Performance Computing Trends*; by Dennis Gannon on *The Information Power Grid*; by Ambuj Goyal on *High Performance Computing - A Ten Year Outlook*; by H. T. Kung on *Computer Network Protocols That Can Guarantee Quality of Service*; by Jay Misra on *A Notation for Hypercubic Computations*; and by Burkard Monien on *Balancing the Load in Networks of Processors*.

I wish to thank the six program vice chairs who worked very closely with me in the process of selecting the papers and preparing the excellent technical program. They are Alok Choudhary, Sajal Das, Vipin Kumar, S. K. Nandy, Mateo Valero, and Pen Yew. Viktor Prasanna and Bhabani Sinha provided excellent feedback about the technical program in their role as general co-chairs. I am grateful to Vipin Kumar for organizing the six excellent keynote speeches. I would like to express my gratitude to Nalini Venkatasubramanian for compiling the proceedings of the conference.

Finally, I also wish to acknowledge the tremendous support provided by my assistant Nancy Singer without whose constant help I could not have performed the job of program chair. She single-handedly did all the work relating to the acknowledgment of the papers, arranging the electronic reviews, collecting the reviews, organizing the

reviews in summary tables for the program committee meeting, and informing all authors of acceptance/rejection decisions.

I wish to welcome all the delegates to the conference and wish them an excellent program.

Prith Banerjee
Program Chair



Conference Organization

GENERAL CO-CHAIRS

Viktor K. Prasanna, University of Southern California
Bhabani P. Sinha, Indian Statistical Institute, Calcutta

VICE GENERAL CHAIR

D.N. Jayasimha, Intel Corporation

PROGRAM CHAIR

Prith Banerjee, Northwestern University, Illinois

PROGRAM VICE CHAIRS

Alok Choudhary, Northwestern University
Sajal Das, University of Texas at Arlington
Vipin Kumar, University of Minnesota
S.K. Nandy, Indian Institute of Science
Mateo Valero, Universidad Politecnica de Catalunya
Pen Yew, University of Minnesota

KEYNOTE CHAIR

Vipin Kumar, University of Minnesota

POSTER/PRESENTATION CHAIR

Sartaj Sahni, University of Florida

PROCEEDINGS CHAIR

Nalini Venkatasubramanian, University of California, Irvine

EXHIBITS CHAIR

R. Govindarajan, Indian Institute of Science

AWARDS CHAIR

Arvind, MIT

TUTORIALS CHAIR

Manavendra Misra, Colorado School of Mines

INDIA CO-ORDINATOR

K. R. Venugopal, University Visvesvaraya College of Engineering

PUBLICITY CHAIR

Kiran Bondalapati, University of Southern California

FINANCE CO-CHAIRS

A.K.P. Nambiar, Software Technology Park, Bangalore
Ajay Gupta, Western Michigan University

PROGRAM COMMITTEE

- P.C.P. Bhatt, Kochi University of Technology, Japan
Rupak Biswas, NASA Ames Research Centre
Lynn Choi, University of California, Irvine
Chitta Ranjan Das, Pennsylvania State University
Ajoy Datta, University of Nevada, Las Vegas
Hank Dietz, Purdue University
Richard Enbody, Michigan State University
Fikret Ercal, University of Missouri, Rolla
Sharad Gavali, NASA
Siddhartha Ghoshal, Indian Institute of Science
Ananth Grama, Purdue University
Manish Gupta, IBM Watson Research Centre
Frank Hsu, Fordham University
Matthew Jacob, Indian Institute of Science
Divyesh Jadav, IBM Almaden Research Centre
Joseph JaJa, University of Maryland
Mahmut Kandemir, Syracuse University
George Karypis, University of Minnesota
Ralph Kohler, Air Force Research Labs
Dilip Krishnaswamy, Intel Corporation
Shashi Kumar, Indian Institute of Technology, New Delhi
Zhiyuan Li, Purdue University
David Lilja, University of Minnesota
Rajib Mall, Indian Institute of Technology, Kharagpur
Nihar Mahapatra, SUNY at Buffalo
Prasant Mohapatra, Iowa State University
Bhagirath Narahari, George Washington University
Stephan Olariu, Old Dominion University
Ajit Pal, Indian Institute of Technology, Kharagpur
Dhabaleswar Panda, The Ohio State University
Cristina Pinotti, IEI-CNR, Italy
Sanguthevar Rajasekaran, University of Florida
J. Ramanujam, Louisiana State University
Abhiram Ranade, Indian Institute of Technology, Mumbai
Pandu Rangan, Indian Institute of Technology, Chennai
Sanjay Ranka, University of Florida
A. L. Narasimha Reddy, Texas A & M University
Amber Roy-Chowdhury, Transarc Corp.
P. Sadayappan, The Ohio State University
Subhash Saini, NASA Ames Research Centre
Sanjeev Saxena, Indian Institute of Technology, Kanpur
Elizabeth Shriver, Bell Labs
Rahul Simha, College of William and Mary
Per Stenstrom, Chalmers University, Sweden
Valerie Taylor, Northwestern University
Rajeev Thakur, Argonne National Lab
Josep Torrellas, University of Illinois, Urbana
Nian-Feng Tzeng, University of Southwestern Louisiana
Albert Y. Zomaya, University of Western Australia

STEERING COMMITTEE

Arvind, MIT

Vijay Bhatkar, C-DAC

Wen-Tsuen Chen, National Tsing Hua University, Taiwan

Yoo Kun Cho, Seoul National University, Korea

Michel Cosnard, Ecole Normale Supérieure de Lyon, France

José Duato, Universidad Politecnica de Valencia, Spain

Ian Foster, Argonne National Labs.

Anoop Gupta, Stanford University and Microsoft Research

Louis Hertzberger, University of Amsterdam, The Netherlands

Chris Jesshope, Massey University, New Zealand

David Kahaner, Asian Technology Information Program, Japan

Guojie Li, National Research Centre for Intelligent Computing Systems, China

Miroslaw Malek, Humboldt University, Germany

Lionel Ni, Michigan State University

Lalit M. Patnaik, Indian Institute of Science

Viktor K. Prasanna, USC, Chair

N. Radhakrishnan, US Army

José Rolim, University of Geneva, Switzerland

Sartaj Sahni, University of Florida

Assaf Schuster, Technion, Israel Institute of Technology, Israel

Vaidy Sunderam, Emory University

Satish Tripathi, University of California, Riverside

David Walker, Oak Ridge National Labs

K.S. Yajnik, Yajnik and Associates

Albert Y. Zomaya, University of Western Australia

NATIONAL ADVISORY COMMITTEE

Alok Aggarwal, IBM Solutions Research Centre, India

R.K. Bagga, DRDL, Hyderabad

N. Balakrishnan, Supercomputer Education and Research Centre, Indian Institute of Science

Ashok Desai, Silicon Graphics Systems (India) Private Ltd.

Kiran Deshpande, Mahindra British Telecom Ltd.

H.K. Kaura, Bhabha Atomic Research Centre

Hans H. Kafka, Siemens Communication Software Ltd.

Ashish Mahadwar, PlanetAsia Ltd.

Pradeep Marwaha, Cray Research International Inc.

Susanta Misra, Motorola India Electronics Ltd.

Som Mittal, Digital Equipment (India) Ltd.

B.V. Naidu, Software Technology Park, Bangalore

N.R. Narayana Murthy, Infosys Technologies Ltd.

S.V. Raghavan, Indian Institute of Technology, Chennai

V. Rajaraman, Jawaharlal Nehru Centre for Advanced Scientific Research

S. Ramadorai, Tata Consultancy Services, Mumbai

K. Ramani, Future Software Pvt. Ltd.

S. Ramani, National Centre for Software Technology

Karthik Ramarao, Hewlett-Packard India Ltd.

Kalyan Rao, Satyam Computers Ltd.

S.B. Rao, Indian Statistical Institute

Uday Shukla, Tata IBM Ltd.

U.N. Sinha, National Aerospace Laboratories

HiPC'99 Reviewers

S. Adlakha	M. Hilgers	S. Rajasekaran
G. Agarwal	F. Hsu	J. Ramanujam
M. Aggarwal	P. Hu	A. Ranade
G. Agrawal	J. Huang	P. Rangan
R. Andonov	M. Jacob	S. Ranka
D. Anvekar	D. Jadav	F. Rastello
R. Badia	J. JaJa	A. Reddy
S. Bandyopadhyay	R. Jayaram	A. Roy-Chowdhury
C. Barrado	K. Kailas	L. Ruan
G. Baumgartner	N. Kakani	P. Sadayappan
W. Bein	M. Kandemir	D. Saha
A. Bhattacharya	G. Karypis	S. Saini
R. Biswas	E. Kim	S. Saxena
P. Calland	R. Kohler	J. Shen
M. Chatterjee	A. Kongmunvattana	L. Shriver
S. Chawla	D. Krishnaswamy	R. Simha
T. Chen	M. Kumar	A. Sivasubramaniam
L. Choi	V. Kumar	J. Sole
A. Choudhary	L. Larmore	S. Song
J. Corbal	S. Latifi	R. Srikant
T. Cortes	P. Lee	P. Stenstrom
C. Das	Z. Li	D. Su
Sajal Das	D. Lilja	S. Subramanya
Samir Das	J. LLosa	P. Tarau
A. Datta	N. Mahapatra	V. Taylor
H. Dietz	R. Mall	R. Thakur
D. Du	G. Manimaran	K. Theobald
R. Enbody	I. Martel	J. Torrellas
F. Ercal	A. Mikler	N. Tzeng
A. Fabbri	P. Mohapatra	A. Vaidya
D. Fernandez	P. Murthy	M. Valero
J. Gabarro	J. Myoupo	Y. Xin
S. Ghoshal	S. Nandy	J. Xue
J. Gonzalez	B. Narahari	P. Yew
R. Govindarajan	S. Nelakuditi	K. Yum
A. Grama	S. Olariu	B. Zheng
M. Gupta	A. Pal	N. Zheng
R. Gupta	D. Panda	A. Zomaya
S. Han	C. Pinotti	

Table of Contents

Session I-A: Architecture/Compilers

Chair: Pradip K. Das, Jadavpur University

Efficient Technique for Overcoming Data Migration in Dynamic Disk Arrays <i>S. Zertal, Versailles University, and C. Timsit, Ecole Supérieure d'Electricité</i>	3
Combining Conditional Constant Propagation and Interprocedural Alias Analysis <i>K. Gopinath and K.S. Nandakumar, Indian Institute of Science, Bangalore</i>	13
Microcaches <i>D. May, D. Page, J. Irwin, and H.L. Muller, University of Bristol</i>	21
Improving Data Value Prediction Accuracy Using Path Correlation <i>W. Mohan and M. Franklin, University of Maryland</i>	28
Performance Benefits of Exploiting Control Independence <i>S. Vadlapatla and M. Franklin, University of Maryland</i>	33
Fast Slicing of Concurrent Programs <i>D. Goswami, Indian Institute of Technology, Kharagpur and R. Mall, Curtin University of Technology</i>	38
Session I-B: Cluster Computing <i>Chair: R. Govindarajan, Indian Institute of Science</i>	
VME Bus-Based Memory Channel Architecture for High Performance Computing <i>M. Sharma, A. Mandal, B.S. Rao, and G. Athithan, Defense Research and Development Organization</i>	45
Evaluation of Data and Request Distribution Policies in Clustered Servers <i>A. Khaleel and A.L.N. Reddy, Texas A & M University</i>	55
Thunderbolt: A Consensus-Based Infrastructure for Loosely-Coupled Cluster Computing <i>H. Praveen, S. Arvindam, and S. Pokarna, Novell Software Development Pvt. Ltd.</i>	61
Harnessing Windows NT for High Performance Computing <i>A. Saha, K. Rajesh, S. Mahajan, P.S. Dhekne, and H.K. Kaura, Bhabha Atomic Research Centre</i>	66

Performance Evaluation of a Load Sharing System on a Cluster of Workstations <i>Y. Hajmehmoud, P. Sens, and B. Folliot, Université Pierre et Marie Curie</i>	71
Modeling Cone-Beam Tomographic Reconstruction Using LogSMP: An Extended LogP Model for Clusters of SMPs <i>D.A. Reimann, Albion College, and V. Chaudhary, and I.K. Sethi, Wayne State University</i>	77
Session II-A: Compilers and Tools <i>Chair: Manoj Franklin, University of Maryland</i>	
A Fission Technique Enabling Parallelization of Imperfectly Nested Loops <i>J. Ju, Pacific Northwest National Laboratory and V. Chaudhary, Wayne State University</i>	87
A Novel Bi-directional Execution Approach to Debugging Distributed Programs <i>R. Mall, Curtin University of Technology</i>	95
Memory-Optimal Evaluation of Expression Trees Involving Large Objects <i>C.-C. Lam, D. Cociorva, G. Baumgartner, and P. Sadayappan, Ohio State University</i>	103
Resource Usage Modelling for Software Pipelining <i>V.J. Ramanan and R. Govindarajan, Indian Institute of Science, Bangalore</i>	111
An Interprocedural Framework for the Data and Loops Partitioning in the SIMD Machines <i>J. Lin, Z. Zhang, R. Qiao, and N. Zhu, Academia Sinica</i>	120
Tiling and Processors Allocation for Three Dimensional Iteration Space <i>H. Bourzoufi, B. Sidi-Boulenouar, and R. Andonov, University of Valenciennes</i>	125
Session II-B: Scheduling <i>Chair: Rajib Mall, Indian Institute of Technology, Kharagpur</i>	
Process Migration Effects on Memory Performance of Multiprocessor Web-Servers <i>P. Foglia, R. Giorgi, and C.A. Prete, Università di Pisa</i>	133
Adaptive Algorithms for Scheduling Static Task Graphs in Dynamic Distributed Systems <i>P. Das, D. Das, and P. Dasgupta, Indian Institute of Technology, Kharagpur</i>	143

Scheduling Strategies for Controlling Resource Contention on Multiprocessor Systems <i>S. Majumdar, Carleton University</i>	151
Deadline Assignment in Multiprocessor-Based Fault-Tolerant Systems <i>S.K. Kodase, N.V. Satyanarayana, A. Pal, Indian Institute of Technology, Kharagpur, and R. Mall, Curtin University of Technology</i>	158
Affinity-Based Self Scheduling for Software Shared Memory Systems <i>W. Shi and Z. Tang, Chinese Academy of Sciences</i>	163
Efficient Algorithms for Delay Bounded Multicast Tree Generation for Multimedia Applications <i>N. Narang, G. Kumar, and C.P. Ravikumar, Indian Institute of Technology, New Delhi</i>	169
Panel	
Whither Indian Computer Science R & D? <i>Moderator: Sriram Vajapeyam, Indian Institute of Science</i>	
Mini Symposium	
High Performance Data Mining <i>Organizers: Vipin Kumar and Jaideep Srivastava, University of Minnesota</i>	
Session III-A: Parallel Algorithms - I	
<i>Chair: Amar Mukherjee, University of Central Florida</i>	
Self-Stabilizing Network Decomposition <i>F. Belkouch, Université de Technology de Compiègne, M. Bui, Université de Paris, L. Chen, Ecole Centrale de Lyon, and A.K. Datta, University of Nevada</i>	181
Performance Analysis of a Parallel PCS Network Simulation <i>A. Boukerche, A. Fabbri, O. Yildiz, University of North Texas, and S.K. Das, University of Texas at Arlington</i>	189
Ultimate Parallel List Ranking? <i>J. F. Sibeyn, Max-Planck-Institut für Informatik</i>	197
A Parallel 3-D Capacitance Extraction Program <i>Y. Yuan and P. Banerjee, Northwestern University</i>	202
Parallel Algorithms for Queries with Aggregate Functions in the Presence of Data Skew <i>Y. Jiang, K.H. Liu, and C.H.C. Leung, Victoria University of Technology</i>	207

A Deterministic On-Line Algorithm for the List-Update Problem <i>H. Mahanta and P. Gupta, Indian Institute of Technology, Kanpur</i>	212
Session III-B: Mobile Computing - I <i>Chair: Sajal Das, University of North Texas</i>	
Link-State Aware Traffic Scheduling for Providing Predictive QoS in Wireless Mobile Multimedia Networks <i>A.Z.M.E. Hossain and V.K. Bhargava, University of Victoria</i>	219
Enhancing Mobile IP Routing Using Active Routers <i>K.W. Chin, M.Kumar, Curtin University of Technology, and C. Farrell, NDG Software</i>	229
Adaptive Scheduling at Mobiles for Wireless Networks with Multiple Priority Traffic and Multiple Transmission Channels <i>S. Damodaran, Cisco Systems and K.M. Sivalingam, Washington State University</i>	234
An Analysis of Routing Techniques for Mobile and Ad Hoc Networks <i>R. V. Boppana, M.K. Marina, and S.P. Konduru, University of Texas at San Antonio</i>	239
MobiDAT: Mobile Data Access and Transactions <i>D. Bansal, M. Kalia, and H. Saran, Indian Institute of Technology, New Delhi</i>	246
Session IV-A: Parallel Algorithms - II <i>Chair: Dilip Krishnaswamy, Intel Corporation</i>	
Optimal k-ary Divide and Conquer Computations on Wormhole 2-D and 3-D Meshes <i>J. Trdlička and P. Tvrďák, Czech Technical University</i>	253
Parallel Real Root Isolation Using the Descartes Method <i>T. Decker and W. Krandick, University of Paderborn</i>	261
Cellular Automata Based Transform Coding for Image Compression <i>K. Paul, Bengal Engineering College, D.R. Choudhury, Indian Institute of Technology, Kharagpur, and P.P. Chaudhuri, Bengal Engineering College</i>	269
A Parallel Branch-and-Bound Algorithm for the Classification Problem <i>S. Balev, R. Andonov, and A. Freville, Université de Valenciennes et du Hainaut-Cambresis</i>	274
Parallel Implementation of Tomographic Reconstruction Algorithms on Bus-Based Extended Hypercube <i>K. Rajan and L.M. Patnaik, Indian Institute of Science, Bangalore</i>	279

An Optimal Hardware-Algorithm for Selection Using a Fixed-Size Parallel Classifier Device <i>S. Olariu, Old Dominion University, M.C. Pinotti, Istituto di Elaborazione della'Informazione, and S.Q. Zheng, University of Texas at Dallas</i>	284
Session IV-B: Mobile Computing - II <i>Chair: Ajit Pal, Indian Institute of Technology, Kharagpur</i>	
A Novel Frame Structure and Call Admission Control for Efficient Resource Management in Next Generation Wireless Networks <i>N.K. Kakani, S.K. Das, University of North Texas, S.K. Sen, Nortel Networks</i>	291
Harmony - A Framework for Providing Quality of Service in Wireless Mobile Computing Environment <i>A. Lele and S.K. Nandy, Indian Institute of Science, Bangalore</i>	299
Stochastic Modeling of TCP/IP over Random Loss Channels <i>A.A. Abouzeid, M. Azizoglu, and S. Roy, University of Washington</i>	309
Accurate Approximate Analysis of Dual-Band GSM Networks with Multimedia Services and Different User Mobility Patterns <i>M. Meo and M.A. Marsan, Politecnico di Torino</i>	315
Paging Strategies for Future Personal Communication Services Network <i>P.S. Bhattacharjee, Telephone Bhawan, D. Saha, Jadavpur University, and A. Mukherjee, Pricewaterhouse Coopers Ltd.</i>	322
Session V-A: Parallel Applications <i>Chair: C.P. Ravikumar, Indian Institute of Technology, Delhi</i>	
A Framework for Matching Applications with Parallel Machines <i>J. In, C. Jin, J. Peir, S. Ranka, and S. Sahni, University of Florida</i>	331
A Parallel Monte Carlo Algorithm for Protein Accessible Surface Area Computation <i>S. Aluru and D. Ranjan, New Mexico State University and N. Futamura, Syracuse University</i>	339
Parallelisation of a Navier-Stokes Code on a Cluster of Workstations <i>V. Ashok and T.C. Babu, Vikram Sarabhai Space Centre</i>	349
I/O Implementation and Evaluation of Parallel Pipelined STAP on High Performance Computers <i>W.-k. Liao, Syracuse University, A. Choudhary, Northwestern University, D. Weiner and P. Varshney, Syracuse University</i>	354

Efficient Parallel Adaptive Finite Element Methods Using Self-Scheduling Data and Computations <i>A.K. Patra, J. Long, and A. Laszloffy, State University of New York at Buffalo</i>	359
Avoiding Conventional Overheads in Parallel Logic Simulation: A New Architecture <i>D. Dalton, University College, Dublin</i>	364
Session V-B: Interconnection Networks	
<i>Chair: Bhargab Bhattacharya, Indian Statistical Institute</i>	
Isomorphic Allocation in k-ary n-cube Systems <i>M. Kang and C. Yu, Information and Communications University</i>	373
Unit-Oriented Communication in Real-Time Multihop Networks <i>S. Balaji, University of Illinois, G. Manimaran, Iowa State University, and C.S.R. Murthy, Indian Institute of Technology, Chennai</i>	381
Counter-Based Routing Policies <i>X. Liu, Y. Xiang, and T.J. Li, Chinese Academy of Sciences</i>	389
Minimizing Lightpath Set-up Times in Wavelength Routed All-Optical Networks <i>M. Shiva Kumar and P.S. Kumar, Indian Institute of Technology, Chennai</i>	394
Design of WDM Networks for Delay-Bound Multicasting <i>C.P. Ravikumar, M. Sharma, and P. Jain, Indian Institute of Technology, New Delhi</i>	399
Generalized Approach towards the Fault Diagnosis in Any Arbitrarily Connected Networks <i>B. Dasgupta, S. Dasgupta, and A. Chowdhury, Jadavpur University</i>	404
Author Index	411

Session I-A

Architecture/Compilers
Chair: Pradip K. Das
Jadavpur University

Efficient Technique for Overcoming Data Migration in Dynamic Disk Arrays

Soraya Zertal¹ and Claude Timsit^{2,1}

¹ PRISM, Versailles University

45, Av. des Etats-Unis

78000 Versailles, France

² SUPELEC

plateau de moulin

91000 Gif-sur-yvette, France

{zertal,timsit}@prism.uvsq.fr

Abstract. We introduce a new concept for storage management on multi-disk systems, called dynamic redundancy. It associates the redundancy level which characterizes the storage organization to sets of data and not to storage areas while minimizing data migration for reconfiguration. The result is a high quality multi-disk storage system, capable to support various storage organizations at once. It provides transformations between them with a minimum of (if any) data migration.

1 Introduction

Improvements in microprocessor performance have greatly outpaced improvements in the I/O one. To overcome this I/O crisis, RAID¹ technology has been proposed ([7][5][4][3][8]). It accommodates growing storage requirements of capacity, performance and reliability. Berkeley classification [3] counts seven levels [0..6]. The most interesting and used ones are : RAID1, RAID3 and RAID5. RAID1 replicates all data, it is suitable for data for which reliability or performance² access requirements are extremely high. RAID3 uses XORparity which decreases the redundancy space cost. It provides an excellent performance for data transfer-intensive applications. RAID5 distributes the XORparity accross all the disks. It performs best for transaction processing applications [1]. So, each RAID level performs well only for relatively narrow range of workloads [2]. Making a wrong RAID level choice leads to poor performance and data loss due to data migration [9]. Thus, the use of a **workload-adaptive storage** is necessary. Up to now, proposed methods realizing this type of storage ([9],[6]) do not suppress this migration. They define two data classes (active/inactive) and separate storage space into two predefined regions (RAID1/RAID5). Active data is stored in the RAID1 region and the inactive data is stored in the RAID5 one. In accordance with their activity, data migrates from a region to the other. We propose a new concept called **dynamic redundancy** which aims to :

— provide **different RAID levels on the same storage system**, without any physical space separation and a complete independence of any specific placement scheme,

¹ Redundant Arrays of Independant Disks.

² Especially in degraded mode, so with one disk failure.

— achieve a **dynamic data layout change** according to workload while minimizing data migration, and even avoiding it in some cases.

In section 2, we describe two existing workload-adaptive storage methods. We present the dynamic redundancy concept [10] in section 3 and its realization strategies in section 4, then we evaluate it in section 5. Simulation procedure and its results are presented in section 6 and 7. Section 8 concludes this paper.

2 Related work

In the **HP autoRAID** [9], each disk is divided into **Physical EXtents** containing a set of segments. PEXes on different disks are grouped to form **Physical Extent Group**. A PEG may be RAID1 or RAID5 configured, or may be free. In the mirrored storage, the write causes a back-end write to the two disks if they are in the mirrored storage, otherwise it causes a migration of some mirrored data to the RAID5 area. In this one, the write appends the data to the current RAID5 write PEG. The inter-PEG migration slows down writes and can lead to data loss. The fixed ratio between the two storage areas intensifies this migration.

In the **Hot mirroring** [6], all disks are physically divided into a RAID1 area and a RAID5 one. RAID5 stripes are made into each column of disks. Mirrored pairs are stored on disks from different columns. This orthogonal placement reduces the performance degradation during data reconstruction periods. Usually, the written blocks are issued on RAID1. Even if the write concerns a cold block, migration of such block is performed from the RAID5 area. If there is not enough free space in the RAID1 area, some mirrored data migrate to the RAID5 area.

3 The Dynamic Redundancy concept

We propose the Dynamic Redundancy to avoid the data migration issued from the physical storage space subdivision into regions with a specific redundancy/data placement scheme. The Dynamic Redundancy concept is totally different. It relies on the association of the redundancy to the stored data and not to the storage space or one of its areas. The main result of giving the redundancy level a relation to the stored data, is a storage space completely independent of any particular data placement pattern. We call it Dynamic Redundancy because the redundancy level of data changes dynamically as time goes on according to its activity. Data blocks are reorganized on the same space with the minimum of (if any) data blocks migration. The storage space reorganization is performed using a mapping table to map logical addresses to the associated redundancy level and the corresponding physical address as well as a temporary buffer. We have performed some simulations to estimate data activity thresholds which indicate at what requests arrival rate and with which request type percentage, it is more suitable to use a specific data layout scheme than another. Thus, at which point of the workload, the layout scheme of the concerned data will be transformed.

4 Realization strategies

We are interested in RAID1, RAID3 and RAID5. Below, we describe the proposed transformation strategies between them and the used parameters.

- E : the set of the storage system disks.
- N : the number of the storage system disks. $\text{card}(E) = N$.
- D : the set of data which is concerned by the layout pattern change.
- DSK_i : the disk marked i from E .
- $gr_sto(D)$: the actual storage group of (D) . It is a set of disks storing the data (D) .

We assume that $\text{card}(\text{gr_sto}(D)) = 2K$ in RAID1 and $(K + 1)$ otherwise.

$\text{gr_sto}_I(D)$: the storage group of (D) in the initial storage organization.

$\text{gr_sto}_T(D)$: the storage group of (D) in the target storage organization.

DSK_iM : the mirrored disk of DSK_i . It is significant in RAID1 if $DSK_i \in \text{gr_sto}(D)$.

n : the number of data segments in (D) . It depends on the striping unit size.

n_i : the number of data segments in (D) , stored on DSK_i from $\text{gr_sto}(D)$.

$\text{Max}_i n_i$: the number of stripes formed by (D) on $\text{gr_sto}(D)$.

$\text{adr_beg}_i(D)$: the absolute begin address of the n_i segments of (D) on DSK_i .

DR : the redundancy disk according to the target organization.

(DSK, adr) : the segment addressed by adr on the DSK disk.

4.1 The RAID1 to RAID3 transformation

This transformation concerns data for which reliability and high access performance needs decrease. The parity-disk on the target organization avoids the high redundancy space cost and performs well for the new profile access. This transformation (figure 1) is performed according to the pseudo code bellow :

1. Choose $DR \in \{E/\text{gr_sto}_I(D)\}$ because of data availability.
2. Read mirrored segments (on DSK_iM). For transparency reasons, the read is performed on the mirrored copy while the original one is still usable.
3. Calculate the target redundancy by an "XOR" function on read segments.
4. Write the target redundancy (XORparity) on the chosen DR .
5. Update the mapping table according to the new XORparity physical address and invalidate the mirrored segments (on DSK_iM).

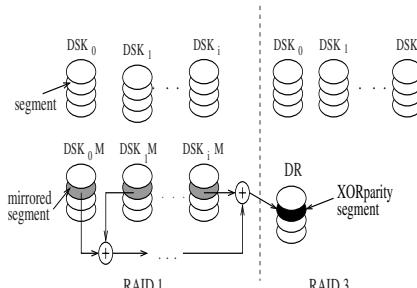


Fig. 1. RAID1-RAID3 transformation

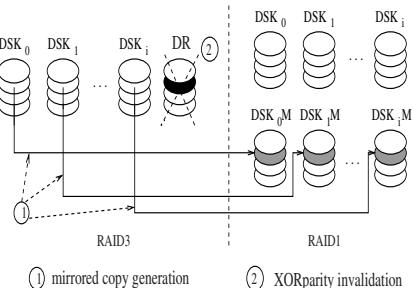


Fig. 2. RAID3-RAID1 transformation

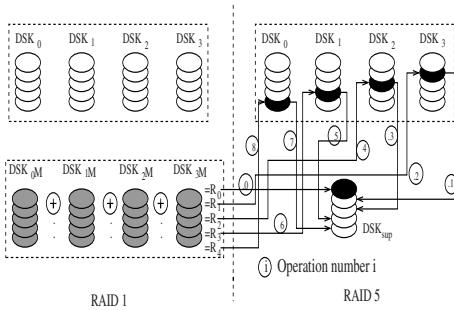
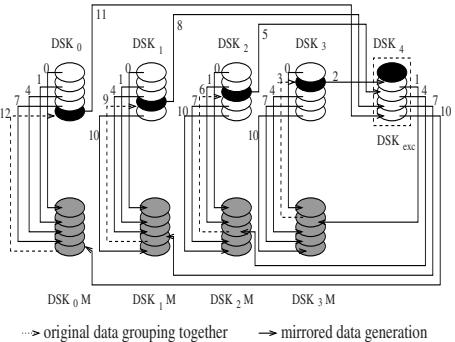
We note above all, that no data migration is used. During all the transformation period, original data is still usable and if there is any disk failure, the reconstruction is guaranteed by mirrored segments still valid until the last step.

4.2 The RAID3 to RAID1 transformation

This transformation concerns revived data which need high performance and extremely high reliability. The transformation operations are performed according to the order shown in figure 2. No migration is used in the RAID3 to RAID1 transformation also. However, the construction of mirrored copy of (D) is based on the original one. Thus, (D) is temporary unavailable for users during the read period. Reconstructing data if there is a disk failure during the transformation period is still possible by the valid XORparity.

4.3 The RAID1 to RAID5 transformation

The transformation of a RAID1 organization into a RAID5 one is justified by the outdated data and their costly storage space on the RAID1 in comparison to their activity. The round-robin distribution of the XORparity provides good

**Fig. 3.** RAID1-RAID5 transformation**Fig. 4.** RAID5-RAID1 transformation

performance for the new commonly transaction processing requests. The transformation (figure 4) is performed according to the algorithm bellow.

1. Choose DSK_{sup} , add it to $\{E/gr_sto_I(D)\}$ to form $gr_sto_T(D)$.
2. Construct the RAID5 stripes:

```

Let num = 0
while (num < Max_i(n_i)) do
    * IdentifyDR: DR = (K - 1) + (A * K) - num with A = [num]
    * Calculate the XORparity:
        XORparity = ⊕_{i=0}^{K-1}(DSK_i, num + adr_beg_i(D)).
        with DSK_i ∈ gr_sto_I(D) + and num < n_i
    * Write the XORparity on DR:
        if (DR ≠ DSK_{sup}) then
            (DSK_{sup}, num + adr_beg_{DSK_{sup}}(D)) ← (DR, num + adr_beg_{DR}(D))
        end if
        (DR, num + adr_beg_{DR}(D)) ← XORparity
end while

```

3. Update the mapping table according to the new XORparity physical address and invalidate the mirrored segments on each $DSK_i M$.

We notice the migration of some data segments, due to the round-robin distribution of the redundancy. When DR coincides with an original data disk, a copy of this original data is stored on DSK_{sup} before writing redundancy which leads to one segment migration. During the transformation period, data availability is guaranteed by the respect of the operations execution order (figure 3).

4.4 The RAID5 to RAID1 transformation

This transformation is justified by the activity requirements of the revived data. The RAID1 fulfills these performance and reliability requirements by the two independent data paths and the reduction of the data reconstruction delay. Such transformation operations are performed according to the order shown in figure 4 which guarantees the data availability during the transformation period. The migration of some segments is inevitable during original data grouping together on $\{DSK_i\}$ and the XORparity transfer on DSK_{exc} .

5 Dynamic Redundancy Evaluation

Our goal being to limit data migration during the storage reorganization, we are interested in three parameters: the amount of migrant data segments and the temporary buffer size (table 1) and the number of write operations (table 2). We evaluate these parameters using the dynamic redundancy method (D.R.) and compare them to those obtained without using it (W.D.R.). For the representation clarity, we consider $expr = Max_i(n_i) - (K + 1)[\frac{Max_i(n_i)}{(K+1)}]$.

Initial-target organizations	migrant segments number		Temporary buffer size (block)
	W.D.R.	D.R.	
RAID1-RAID3			$K * \text{Max}_i(n_i)$
RAID3-RAID1	n	0	$K * \text{Max}_i(n_i)$
RAID1-RAID5			$\frac{\text{Max}_i(n_i)}{K+1} * K(K+2)$ if $\text{expr} = 0$ $\frac{\text{Max}_i(n_i)}{K+1} * K(K+2) + K^*$ if $\text{expr} = 1$
RAID5-RAID1	n	$K[\frac{\text{Max}_i(n_i)}{K+1}]$ if $\text{expr} = 0$ $K[\frac{\text{Max}_i(n_i)}{K+1}] + \text{expr} - 1$ else	$(\text{Max}_i(n_i) - \text{expr} - 1)(K + 1)$ else

Table 1. Migrant segments evaluation

Initial-target organizations	write operations number	
	W.D.R.	D.R.
RAID1-RAID3	$n + \text{Max}_i(n_i)$	$\text{Max}_i(n_i)$
RAID3-RAID1	$2n$	n
RAID1-RAID5	$n + \text{Max}_i(n_i)$	$(2K + 1)[\frac{\text{Max}_i(n_i)}{K+1}]$ if $\text{expr} = 0$ $(2K + 1)[\frac{\text{Max}_i(n_i)}{K+1}] + 1 + 2 * (\text{expr} - 1)$ else
RAID5-RAID1	$2n$	$K(K + 3)[\frac{\text{Max}_i(n_i)}{K+1}]$ if $\text{expr} = 0$ $K(K + 3)[\frac{\text{Max}_i(n_i)}{K+1}] + k + (K + 2) * (\text{expr} - 1)$ else

Table 2. Write operations evaluation

6 Simulation

Simulations were performed to measure the benefit of using the Dynamic Redundancy method for overcomming data migration, thus improving the storage quality. These simulations are in accordance with the three steps bellow :

- The calculation of data activity thresholds according to different configurations and parameters (figure 5 and 6) for which the performance of the three RAID levels diverge (fig. 7 to 15). The goal being to determine with which workload configuration, it is better to use a specific RAID level.
- The study of the storage system behaviour using two layout schemes (RAID1/ RAID5) on separate physical storage areas and its impact on the response time (figure 16 and 17).
- The study of the storage system behaviour using the Dynamic Redundancy method which offers the possibility of using multiple layout schemes at once on the same storage space (fig. 16 and 17). Transformations between such storage organizations are performed according to the calculated thresholds and using algorithms described in section 4.

For this, we have developped a full event driven hardware simulator for the RAID system which is parametrized by the RAID configuration and all the disk and other component characteristics.

7 Results discussion

On figures (7 to 9), RAID1 gives best user response time, RAID3 and RAID5 are almost similar because there is no write requests. Thus the RAID5 write penalty is avoided. We also note that the increase of the response time is less important after the request arrival rate of 50 req/s because the wait time becomes the most important component of the response time. As the request size becomes larger, RAID3 and RAID5 become closer to each other. This is due to the equivalence between req/IO and the Byte/s throughputs for such request size. On figures (10 and 15), the gap between the RAID3 and the RAID5 becomes more significant.

Disk capacity (MB)	2547
Disk geometry (cyl.)	2756
Disk average seek time (ms)	8
Disk rotaional speed (tr/mn)	7200
Disk extern / intern transfer rate (MBytes/s)	10 / 36
Requests size (KB)	64 256 512
Requests inter-arrival time (req/s)	[5 . . 100]
Access adresses	uniformly distributed over data
Number of disks	6
Stripe unit size (KB)	4
Stripe wide	RAID1 : 2 stripe-units RAID3/RAID5 : 6 stripe-units
RAID5 data layout	Left-Symmetric

Fig.5. Simulation parameters

When the write requests percentage of the considered workload increase, this appears at small requests arrival rates. For example, this is typically true at 20 req/s for 33% of write requests and respectively at 17 req/s , 12 req/s and 8 req/s for 50% , 66% and 100% write requests. For the simulated workload configurations, the user response time for RAID3 and RAID5 reaches closely high values when requests arrival rates are small. RAID1 is considered the most interesting storage organization when the request arrival rate is 14 req/s for 33% write requests, respectively when it is 11 req/s , 9 req/s and 6 req/s for 50% , 66% and 100% of write requests. The RAID3 and the RAID5 are very close, which is justified by the large requests in the simulated configurations. Such requests do not refine the analysis of the RAID3 and the RAID5 but are representative of image processing workloads.

We have investigated the mean response time behaviour in both two storage systems (W.D.R, D.R) corresponding to the second and the third simulation steps. We performed them using workload configurations 7 and 9. Results of this investigation are compared as shown on figures 16 and 17. We notice the improvement of the user response time in both systems. The few (if any) amount of data migration using D.R. makes this improvement more significant. In fact, the mean response time has the same trend of that obtained using a RAID1. Finally, we give a simple histogram on figure 18 which shows clearly the percentage of requests leading to data migration. Such percentage is proportional to the write requests in the used workload. Using W.D.R approach, 32% (respectively 8% using D.R) of requests lead to migration when workload is composed at 50% of write requests and 80% (respectively 13% using D.R) when the workload contains only write requests. The migration percentage drops when using D.R because the frequent choice of RAID3 as the target organization. This choice is justified by the nearby RAID3 and RAID5 performance for such large requests.

8 Conclusion

In this paper, we have presented the new concept of Dynamic Redundancy which associates the redundancy to the data and not to the physical storage space and minimize data migration. Both validation and simulation results confirm the benefit of its use in multi-disks storage systems to improve their quality. The used workload configurations are representative of the image processing applications. We have refined our investigations in comparing RAID3 and RAID5 using new and more appropriate workload configurations. In fact, workloads composed of small requests corresponding to those issued from OLTP applications leads to a threshold for which the RAID3 and the RAID5 performance are different. In the mean time, we continue our studies on the storage customization by applying

Config.	Write percentage	Request size
1	0	64 KB
2	0	256KB
3	0	512KB
4	33	64 KB
5	33	256 KB
6	33	512 KB
7	50	64 KB
8	66	64 KB
9	100	64 KB

Fig.6. Used configurations

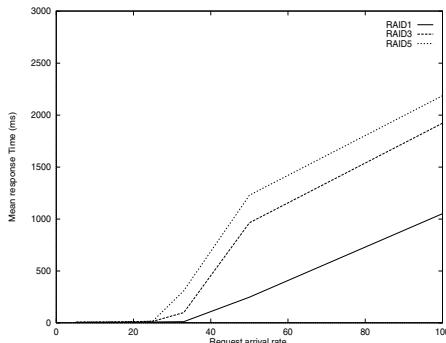


Fig. 7. Configuration 1

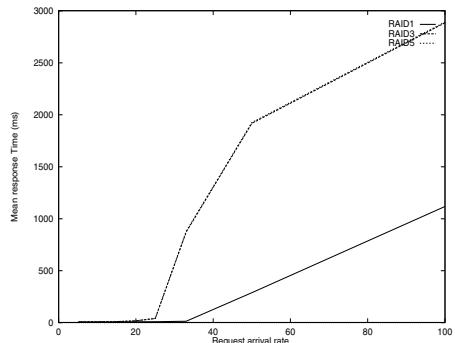


Fig. 8. Configuration 2

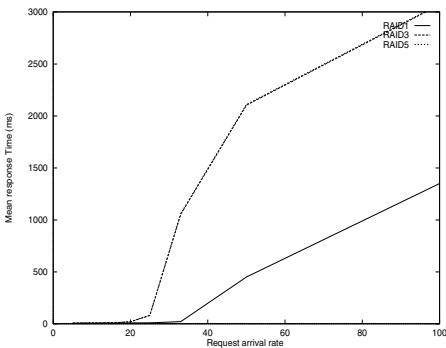


Fig. 9. Configuration 3

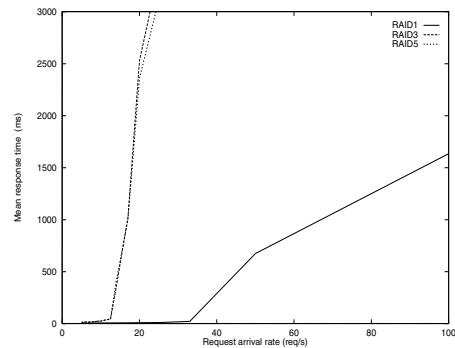


Fig. 10. Configuration 4

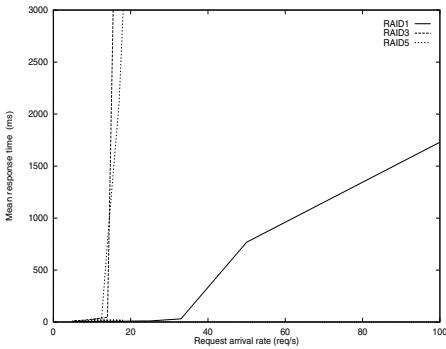


Fig. 11. Configuration 5

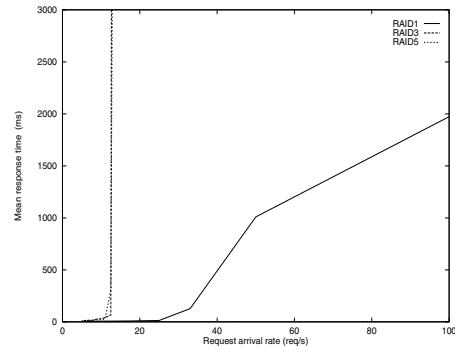


Fig. 12. Configuration 6

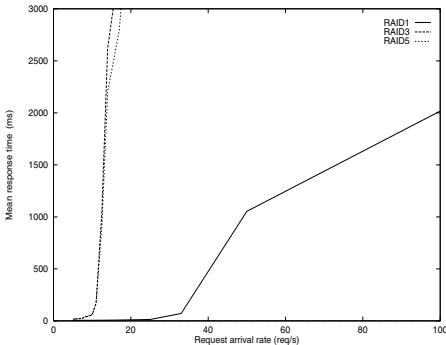


Fig. 13. Configuration 7

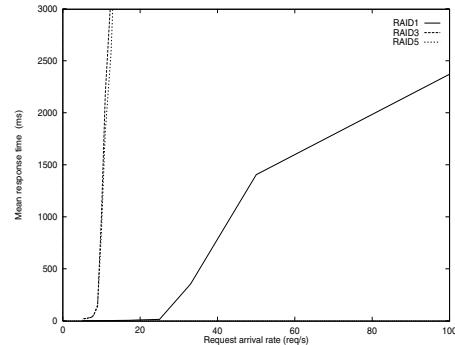


Fig. 14. Configuration 8

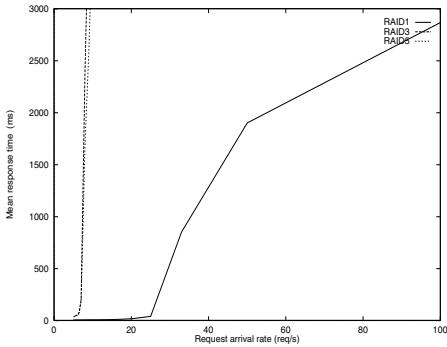


Fig. 15. Configuration 9

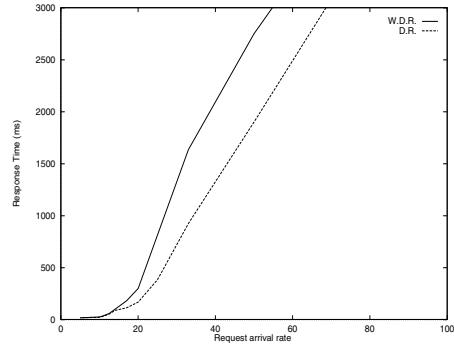


Fig. 16. W.D.R./D.R. response time

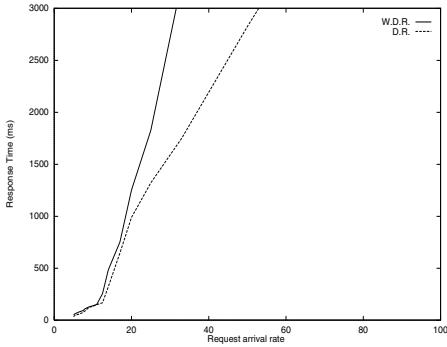


Fig. 17. W.D.R./D.R. response time

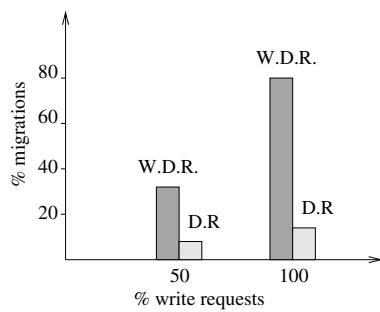


Fig. 18. The migration percentage

different mechanisms such as stripe restriction and extension [11] in order to provide a higher storage quality.

References

1. The RAID Advisory. *The RAIDBOOK : A source Book for RAID Technology*. Lino Lakes, Minn., Jun 1993.
2. P. M. Chen and E. K. Lee. Striping in a raid level-5 disk array. Technical Report CSE-TR-181-93, University of Michigan, 1993.
3. G. Gibson D. A. Patterson and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *ACM sigmod*, Jun 1988.
4. G. Gibson D. A. Patterson, P. M. Chen and R. H. Katz. Introduction to redundant arrays of inexpensive disks (raid). In *IEEE compcon*, 1989.
5. S. Khoshafian M. Livny and H. Boral. Multi-disk management algorithms. In *SIGMETRICS*, May 1987.
6. K. Mogi and M. Kitsuregawa. Hot mirroring : A method of hiding parity update penalty and degradation during rebuilds for RAID5. *ACM Sigmod conf.*, 6, Jun 1996.
7. G. A. Gibson R. H. Katz and D. A. Patterson. Disk system architecture for high performance computing. In *IEEE*, volume 77, Dec 1989.
8. K. Salem and G. Molina. Disk striping. In *IEEE Data Engineering*, Feb 1986.
9. J. Wilkes, R. Golding, and C. Staelin. The HP-AutoRAID hierarchical storage system. *Proc. of 15th ACM Symp. on Operating Systems Principles*, December 1995.
10. S. Zertal. Dynamic redundancy for an efficient storage customization. *Proceedings of ISCIS'98*, Oct 1998.
11. S. Zertal and C. Timsit. Refining the storage customization with the stripe extension mechanism. *Proceedings of ISCIS'99*, Oct 1999.

Combining Conditional Constant Propagation and Interprocedural Alias Analysis

K. Gopinath and K.S. Nandakumar*

Computer Science & Automation
Indian Institute of Science, Bangalore

Abstract: Good quality information from data flow analysis is a prerequisite for code improvement and, in parallelizing compilers, parallelism detection. Typically, compilers do many kinds of data flow analyses and optimizations one after another, which may pose a *phase-ordering* problem: the analyses may have to be run several times until a fixed point is reached. However, when two related optimizations/analyses are performed in combination, it may be possible to obtain better results than iterating over the two analyses independently. Combining optimizations/analyses may be beneficial while potentially enforcing little calculation overhead. This paper discusses combining interprocedural conditional constant propagation (CCP) and interprocedural alias analysis (AA) in pointer-supporting languages.

1 Introduction

Good quality information from data flow analysis is a prerequisite for code improvement and parallelism detection in various code optimizations. Typically, compilers do many kinds of data flow analyses and optimizations one after another, which may pose a *phase-ordering* problem. The compiler has to decide the sequence in which to perform these analyses as different sequences may give different results. Running one analysis may provide information for the other and vice versa. To make best use of this information, the analyses may have to be run several times until a fixed point is reached. However, when two related optimizations/analyses are performed in combination, it may be possible to obtain better results than iterating over the two analyses independently. This might happen as the two optimizations/analyses may feed each other more precise information when run together. In other words, combining optimizations/analyses may be beneficial while potentially introducing only a small overhead. Combining reachability and simple constant propagation thus to result in conditional constant propagation was first demonstrated by Click and Cooper[CC94].

A compiler needs to do pointer or aliasing analysis in order to find out which memory locations a pointer could point to. In the absence of information regarding pointers, a compiler is forced to make the worst case assumption that a pointer could affect any memory location in the program. This is overly conservative and affects every analysis including live-variable analysis, constant propagation, common subexpression elimination, induction variable elimination, dependence analysis, copy propagation and array range-checking.

* Author for correspondence:gopi@csa.iisc.ernet.in

Alias analysis is typically done before any other data flow analysis or optimizations. This means that alias analysis will not be able to reap benefit from conditional constant propagation. If we could combine conditional constant propagation with interprocedural pointer analysis, we could get better solutions to both analyses by arranging for feedback between them.

Use of alias information alone is not sufficient to successfully combine alias analysis and constant propagation. We also need to do *interprocedural side effect* analysis¹ (which is normally done after alias analysis) as we run our combined algorithm.

In this paper, we develop a framework in which interprocedural alias analysis (AA), conditional constant propagation (CCP) and interprocedural side effect analysis can be combined. In the discussions below, we assume that the reader is familiar with the work of Click and Cooper[CC94], Cytron and Gershbein[CG93] and Choi et.al.[CBC93] as we will use their notations and conventions with minor modifications.

2 Combining CCP and AA

2.1 Introduction

Combining two analyses is profitable when the two interact [CC94]. If the results of one data flow analysis problem is used by another but not vice versa, the two analyses can be phase-ordered. However, when they interact, it is *possible* to get a better solution when they are run in combination, each making use of the data provided by the other. CCP is an example of such a combined algorithm: it combines constant propagation with unreachable code elimination thereby providing mutual benefit to the individual optimizations. However, alias and side-effect information is needed beforehand. Thus, alias analysis and side-effect computation do not benefit from CCP. The aim of combining CCP and AA is to enable alias analysis and side-effect analysis to take advantage of CCP and vice versa.

In this paper, we use Wegman and Zadeck's CCP algorithm [WZ91] as a basis to derive a combined algorithm for interprocedural constant propagation and alias analysis (pointer-induced) which is *more powerful* than the two applied individually. This is achieved by using alias information to refine the constant propagation solution and constant propagation to refine the alias solution while running the analyses in tandem rather than in sequence.

2.2 Motivation

Consider Figure 1 for an illustration on how the combined algorithm can give better results. The algorithm makes use of the fact that the value of ‘c’ at conditional S3 is the constant ‘5’ to detect (by evaluating the conditional) that

¹ Interprocedural side effect analysis determines which memory objects are potentially modified by each statement in the program

the false arm of the conditional (statement S5) will not be executed. Hence the alias $\langle *p, y \rangle$ is never generated and will not hold at S6, S7 and S8. The value of ‘y’ at S8 is 6 (from S1) and is not affected by the indirect assignment at S6 since the alias $\langle *p, y \rangle$ does not hold. Thus constant propagation has trimmed the alias sets at S6, S7 and S8 (because of $\langle *p, y \rangle$ not holding) which in turn has enabled ‘y’ at conditional S8 to be determined to be a constant.

If constant propagation is preceded by alias analysis, as alias analysis takes all paths into account, S3 evaluating to a constant is missed out and the generated alias $\langle *p, y \rangle$ pulls down the value of ‘y’ at S6. This solution, however, is safe. Tables 1 and 2 show the solutions when alias analysis and conditional constant propagation are run separately and together.

```

S1:    y = 6;
S2:    c = 5;
      ...
S3:    if (c != 0)  // [c = 5] from S2
S4:        p = &x;
      else
S5:        p = &y;
      ...
S6:    *p = 5; // modification to x not y
S7:    while (P) {
      ...
S8:        if (y == 5) // [y = 6] from S1
S9:            d = 7; // [d = T] as node non-executable
}

```

Fig. 1. Motivating example for combined algorithm

Table 1. Alias solutions for combined vs separate analyses

Alias sets	combined	separate
S1	{}	{}
S2	{}	{}
S3	{}	{}
S4	{ $\langle *p, x \rangle$ }	{ $\langle *p, x \rangle$ }
S5	{}	{ $\langle *p, y \rangle$ }
S6	{ $\langle *p, x \rangle$ }	{ $\langle *p, x \rangle, \langle *p, y \rangle$ }
S7	{ $\langle *p, x \rangle$ }	{ $\langle *p, x \rangle, \langle *p, y \rangle$ }
S8	{}	{ $\langle *p, x \rangle, \langle *p, y \rangle$ }
S9	{}	{ $\langle *p, x \rangle, \langle *p, y \rangle$ }

Table 2. Final values for combined vs separate analyses

Values	combined	separate
c at S3	5	5
y at S8	6	\perp
d at S9	\top	7

However, ignoring inefficiencies, this example can be solved by iterating through CCP, then AA, and again CCP to find y as constant. If, however, we move the loop to surround both the aliasing and the constant assignments, then no amount of iteration will find y as constant. This last aspect is related to the problem of *strongly connected components*(SCC) [PS89] that arise when a data flow solution is incrementally updated due to a change in the flow graph. As it is related to the question of optimistic vs pessimistic algorithms, we discuss it here briefly.

Consider an optimistic CP algorithm that starts out with the assumption that everything is potentially a constant and tries to prove each assumption correct. A *pessimistic* algorithm on the other hand assumes that each variable is \perp and tries to raise each value to a constant if *all* values reaching a node are the same. Pessimistic algorithms do not propagate a value at a use until all the edges reaching that use have been processed. Pessimistic algorithms have the advantage that they can be stopped at any time and still produce a correct though conservative result. Optimistic algorithms, on the other hand, can give incorrect results if stopped before they terminate[CK95].

```
i ← 1;
while (...){
    j ← i;
    i ← f(...);
    ... /* no stores of j here */
    i ← j;
}
```

Fig. 2. Optimistic vs Pessimistic approach

Optimistic algorithms can find more constants than pessimistic approaches because the latter cannot find certain constants in loops [WZ91]. For example, in figure 2 the variable i has the value 1 at the bottom of the loop and this fact is not discovered by pessimistic algorithms because they start with the conservative assumption that i is \perp . SCC, which is an optimistic algorithm, discovers this fact. In other words, pessimistic algorithms do not find the *maximal fixed point* [WZ91].

2.3 Outline of Algorithm

Given monotonic constant propagation and alias analysis frameworks, a combined algorithm for these frameworks needs augmentation with cross-functions. These cross-functions transmit knowledge of non-executable branches (identified by CCP) to alias analysis and CCP would use the alias information in identifying non-executable branches. If the cross-functions transferring information between these two frameworks are monotonic, the combined framework is monotonic and has a greatest fixed point which can be obtained by an iterative worklist technique [CC94].

We need to ensure that the constant propagation algorithm is monotonic; i.e., values of variables start from \top and “fall down” the constant propagation lattice. This implies that we do not need to update the alias and constant solutions as branches marked executable will not be marked later as non-executable: the algorithm is *optimistic*.

We use the static single assignment graph (SSA) as the intermediate form as it forms the basis for many powerful optimizations including constant propagation. In SSA form, alias information and side effect information is represented explicitly by adding additional nodes to the SSA graph. SSA form tries to make the alias information explicit in the program representation by inserting *IsAlias* nodes whenever a variable is potentially modified by an indirect assignment. However, the inclusion of the *entire* alias solution in the graph can result in a tremendous growth in the SSA graph. In the worst case, the program will expand by a factor of V , where V is the number of variables in the program [WZ91]. Empirical results indicate that there will be quite a large number of alias pairs reported at every flow graph node. For example, Landi [LR92] reports that when his aliasing analysis algorithm was run on the program make.c, the obtained solution had an average of 675 may-alias relations per node with a maximum of 2000 relations at one node.

This space explosion can be managed by inserting *IsAlias* nodes depending on the needs of conditional constant propagation in a *demand-driven fashion* [CG93]. As an example, in figure 1, we would need to insert an *IsAlias* node for $< *p, y >$ at S6 *only if* it holds at S6 and there is a use of y (S8) which needs to know if the indirect assignment at S6 modifies y . This controlled inclusion of *may-alias* information in the SSA graph has the additional benefit that optimizations can now proceed faster as they do not have to look at many unnecessary *IsAlias* nodes. Thus, an additional aim of the algorithm is to include only relevant alias information in the SSA graph as necessitated by constant propagation².

We use Cytron et al [CG93]’s technique of incrementally including may-alias information in a SSA graph (explained in the full paper but omitted here) depending on the needs of a particular optimization. Cytron assumes that the entire alias information is available beforehand and he uses simple constant propagation as an example optimization. All paths through a procedure are considered

² We do calculate alias information at every program point. It is the explicit representation of the alias information in the SSA graph which is controlled.

executable (no trimming of paths is considered). In our case, we have to take into account non-executable edges discovered by CCP in the control flow graph. Additionally, full alias information is not available beforehand but is computed as the algorithm proceeds. We can still use his technique after suitable modification and ensure that *IsAlias* nodes are added incrementally in a demand-driven fashion.

The combined algorithm can be thought of as finding a sequence of partial SSA forms $SSA_1, SSA_2, \dots, SSA_\sigma$ where each SSA_{i+1} contains more information than SSA_i . We define SSA_0 as the program in SSA form with the must-alias information already injected and with all edges except the start edge marked as non-executable. Given SSA_i the algorithm processes each executable edge of SSA_i once to give SSA_{i+1} . In this transition, the values of the alias sets are updated and depending on the needs of constant propagation *IsAlias*, *Mod* and ϕ nodes are added. Some edges which were non-executable in SSA_i might be marked as executable in SSA_{i+1} . The process continues until $SSA_\sigma = SSA_{\sigma+1}$, the equality being true when the two SSA forms agree on the alias sets, side effect information at every program point, the executable status of the edges and the values of variables.

Computing alias information along with CCP implies that interprocedural side effect information too is to be computed as we go along as it is not available beforehand. Hence the revised algorithm first does alias analysis and constant propagation in tandem iteratively over the program flow graph. A fixed point is reached when the alias sets at every program point and the values of all variables stabilize. Side effect information is next computed (suitably incrementalized). This may cause the values of certain variables to be lowered causing additional branches to become executable. To take into account these newly marked executable branches, alias analysis and constant propagation is reiterated until a fixed point is reached with the alias, constants and side effect information showing no change.

Before running this algorithm, we need to calculate must-alias information³ and update the SSA graph with this information. This is needed as must-alias information introduces killing assignments which can cause constant propagation to be non-monotonic.

Cross-functions in Combined Method We associate 3 data flow equations with each statement - one for computing the reachability status, R_i , of the node (i.e. executable or non-executable), one for the alias sets, A_i , holding after that node and one for constant propagation. The reachability equations output values from the reachability lattice consisting of two elements $\{\mathcal{U}, \mathcal{R}\}$, where \mathcal{U} denotes that a node is unreachable and \mathcal{R} denotes that it is reachable. Initially all nodes except the start node are unreachable. The constant propagation equations work on the constant propagation lattice, L_c . The lattice for alias analysis is the powerset of the set of all possible aliases in the program. The set of object names

³ under the assumption that all *realizable* paths in the program are taken.

in the program is finite (because of k -limiting) and hence the set of aliases is finite. This ensures that the lattice is bounded.

We have cross-functions which take values from reachability to L_a (\bowtie), from reachability to L_c (\Rightarrow), from L_c to reachability (\preceq) and finally from L_a to L_c . These cross-functions, which are monotonic, make explicit the interaction between the analyses. Table 3 says that an unreachable node always contributes an empty alias set whereas a reachable node allows aliases to flow through. Table 4 does the same for variable values except that unreachable nodes contribute \top .

Table 3. $\bowtie: L_r \times L_a \rightarrow L_a$

\bowtie	A
U	$\{\}$
R	A

Table 4. $\Rightarrow: L_r \times L_c \rightarrow L_c$

\Rightarrow	\perp	C_i	\top
U	\top	\top	\top
R	\perp	C_i	\top

Table 5. $\preceq: L_c \times \mathcal{B} \rightarrow L_r$

\preceq	F	T
T	U	U
F	R	U
T	U	R
\perp	R	R

The \preceq operator (Table 5) is used to compare the value of a predicate with the truth values T and F and the result is mapped onto a value in the reachability lattice. Conditional constant propagation evaluates the predicate; the \preceq operator indicates whether a node is dead or not by mapping the predicate value onto the reachability lattice. This knowledge is transmitted to alias analysis by the \bowtie operator. The operators $+$ and \cdot stand for ‘or’ and ‘and’ respectively and are defined in Tables 7 and 6.

3 Conclusion

We have presented a summary of an algorithm which combines interprocedural constant propagation, interprocedural alias analysis and side effect analysis with demand-driven insertion of alias information in the SSA graph. Here we have not described how the interprocedural aspect relating to alias analysis is handled. Lack of space prevents full description of the algorithm but the main outline has been presented. It can be extended further to include inlining.

Table 6. . : $L_r \times L_r \rightarrow L_r$

[.	\mathcal{U}	\mathcal{R}]
\mathcal{U}	\mathcal{U}	\mathcal{U}		
\mathcal{R}	\mathcal{U}	\mathcal{R}		

Table 7. + : $L_r \times L_r \rightarrow L_r$

[+	\mathcal{U}	\mathcal{R}]
\mathcal{U}	\mathcal{U}	\mathcal{U}	\mathcal{R}	
\mathcal{R}	\mathcal{R}	\mathcal{R}	\mathcal{R}	

As part of future work, it would be useful to undertake an empirical study that would attempt to address the following questions: 1: How many conditionals are found to evaluate to constants? 2: When conditionals evaluate to constants, how does the improved alias/constants solution affect each other and other optimizations?

References

- [CBC93] Jong-Deok Choi, Michael Burke, and Paul Carini. Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 232–245, Charleston, South Carolina, January 1993.
- [CC94] Cliff Click and Keith Cooper. Combining Analyses, Combining Optimizations. TOPLAS, No. 2, '95, pg. 181.
- [CK95] Cliff Click. Combining Analyses, Combining Optimizations. PhD thesis, Computer Science Dept., Rice University, 1995.
- [CFR⁺91] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, October 1991.
- [CG93] Ron Cytron and Reid Gershbein. Efficient accomodation of may-alias information in SSA form. *SIGPLAN Notices*, 28(6):36–45, June 1993.
- [LR92] William Landi and Barbara G. Ryder. A safe approximate algorithm for interprocedural pointer aliasing. *SIGPLAN Notices*, 27(7):235–248, July 1992.
- [LRZ93] William Landi, Barbara G. Ryder, and Sean Zhang. Interprocedural modification side effect analysis with pointer aliasing. *SIGPLAN Notices*, 28(6):56–67, June 1993.
- [MEH93] Rakesh Ghiya Maryam Emami and Laurie J. Hendren. Context-sensitive interprocedural points-to analysis in the presence of pointers. Acaps technical memo 54, School of Computer Science, McGill University, 1993.
- [PS89] Lori L. Pollock and Mary L. Soffa. An incremental version of iterative data flow analysis. *IEEE Transactions on Software Engineering*, SE-15(12), December 1989. Incremental Data Flow Analysis.
- [WZ91] Mark N. Wegman and F. Kenneth Zadeck. Constant propagation with conditional branches. *TOPLAS*, 13(2):181–210, April 1991.

Microcaches

David May, Dan Page, James Irwin, and Henk L. Muller

Department of Computer Science, University of Bristol, UK

<http://www.cs.bris.ac.uk/>

Abstract. We describe a radically new cache architecture and demonstrate that it offers a huge reduction in cache cost, size and power consumption whilst maintaining performance on a wide range of programs. We achieve this by giving the compiler control of the cache and by allowing regions of the cache to be allocated to specific program objects. Our approach has widespread application, especially in media processing and scientific computing.

1 Microprocessor Caches

Current computer architectures rely heavily on the use of cache memory to enable the processor to operate at high speed. Cache management hardware takes no account of the characteristics of specific programs, and in many simple cases performs very inefficiently. An obvious example of this is in the copying program

```
for i=0 to 500 { a[i] = b[i] ; }
```

Here a large region of the cache will be taken over by data which is used only once and may interfere with other objects (instruction blocks or data) competing for the same space.

We propose to give the compiler direct control of the cache. A conventional cache is replaced with a *microcache*, organised so that cache *partitions* can be allocated to data objects by the compiler. Given suitable compiler technology, a microcache can provide the same performance as a conventional data cache many times larger. This paper outlines our basic strategy; a full version is available on-line [1].

2 The Partitioned Microcache

A simple form of partitionable microcache has 2^n lines and can be divided into at most 2^p partitions each of size $2^{(n-p)}$ lines. It can also be divided into partitions of size $2^{(n-x)}$ where $x < p$, or into any combination of different size partitions. Each partition P has an address Pa which corresponds to the address of the first line in the partition, so that an address a within partition P has address $Pa||a$. The address of the line to be used within the partition must, of course, be derived from the memory address used to load or store data.

For optimal results, the addresses used to access successive items in a data structure must be mapped on to different lines within the cache partition used for the structure. If the compiler cannot derive much in the way of maximising reuse or persistence then

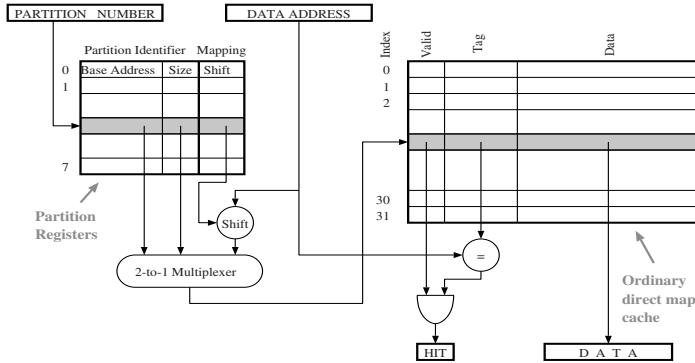


Fig. 1. Possible implementation of a partitioned cache

the result is simply the removal of object interference. This interference removal is the natural product of segregating accesses to data objects by partitioning. However, we are more interested here in cases where the compiler can analyse the access pattern and derive from it the required partition size and the mapping function needed. This function should map addresses on to cache lines efficiently, maximising persistence for a given partition size.

2.1 The Hardware/Software Interface

We pass information from the compiler to the hardware by using an extra parameter to the load and store instruction. This extra parameter supplies the partition information for the memory operation. There are two alternatives which do not require the instruction set to be modified. We can use the high order bits of the address space to contain the partition information, or we could store the “current partition information” in the cache, and change this partition information as and when required. The last solution is only of use if we expect a series of requests to several data objects all to use the same partition. Using higher order address bits is particularly useful when we want to use a microcache together with an existing processor core. It is also suitable for languages like C, where the partition information will be carried along implicitly with a pointer.

The partition information is held in a set of partition control registers that store *partition identifiers* and *mapping information*. The *partition identifier* can be represented by the partition address and partition size. The *mapping information* defines how to hash the addresses of data objects using the partition. In our simple scheme, this information consists of a shift to be applied, but a more complicated scheme might require an XOR of some parts of the address.

In addition to a RISC load/store style instruction set, we provide extra instructions to manage partitions within the cache.

2.2 Implementation

The partitioned cache implementation can be based on a conventional direct mapped cache. A block diagram of our implementation is shown in Figure 1. Addressing the

partition control registers can be pipelined with the execution of the load/store instruction if we assume that the partition operand is a constant parameter of the load and store instructions¹. Also, the partition control registers can simply be general purpose registers.

The partition control register set can be very small indeed. We will need one register for each partition, and in all of our examples twenty partitions are sufficient. The base address and select bits can be stored in $\log_2 l$ bits, where l is the number of lines in the cache. We can combine these in one word of size $1 + \log_2 l$ bits if we wish. Finally, the shift needs to be stored in at most 6 bits.

2.3 Compiling for the Microcache

The compiler takes a program with scalars and (multi-dimensional) arrays, and generates code which includes all of the partitioning information. The compiler performs common optimisations and register allocation in order to keep all scalars and intermediate results in registers. The compiler calculates the required partition sizes, and analyses access patterns in order to determine the required persistence in each of the partitions. We outline this process below, discussing the data object partitioning only. It is also possible to partition the instruction cache (using appropriate metrics) but this is outside the scope of this paper.

Considering only a von-Neumann, imperative programming basis A user application is a set of definitions of variables and then a sequence of statements that operate on those variables. Wherever a variable name appears in a statement of the application, the variable is *referenced*. A variable may be referenced to load its value, or to store a new value into the variable.

Each variable has a set of references associated with it. A reference may carry offset information, for example $a[i, k]$ has $[i, k]$ with i and k being loop counters. Scalars have no offset information.

Using the offset information of references and the variable declaration three properties are computed for each reference: the *stride*, the *group* and the *window*. The *stride* of a reference is simply the distance between successive accesses to a variable when the variable is referenced from within a loop.

The purpose of a *group* is to collect all references to a variable that have an equal stride. As there may be several different strides associated with each variable, each variable may appear in several groups. Scalars always have one group (for their stride is zero by definition).

Each group will be assigned a cache partition. This means accesses to an array may be routed via two or more partitions: if the array is used with two different strides, two partitions will be used. The resulting advantage is two streams of accesses not interfering with each other. However, accessing an array through multiple partitions gives rise to the possibility of inconsistency if the array is updated. To avoid such problems, we can either combine the groups that may give rise to an inconsistency; or issue explicit consistency directives to the cache partitions in question (the consistency overhead is very small, requiring a single cycle per partition that needs updating). We have tried both techniques

¹ This assumption is not true for systems that require (for example) dynamic partitioning within a thread

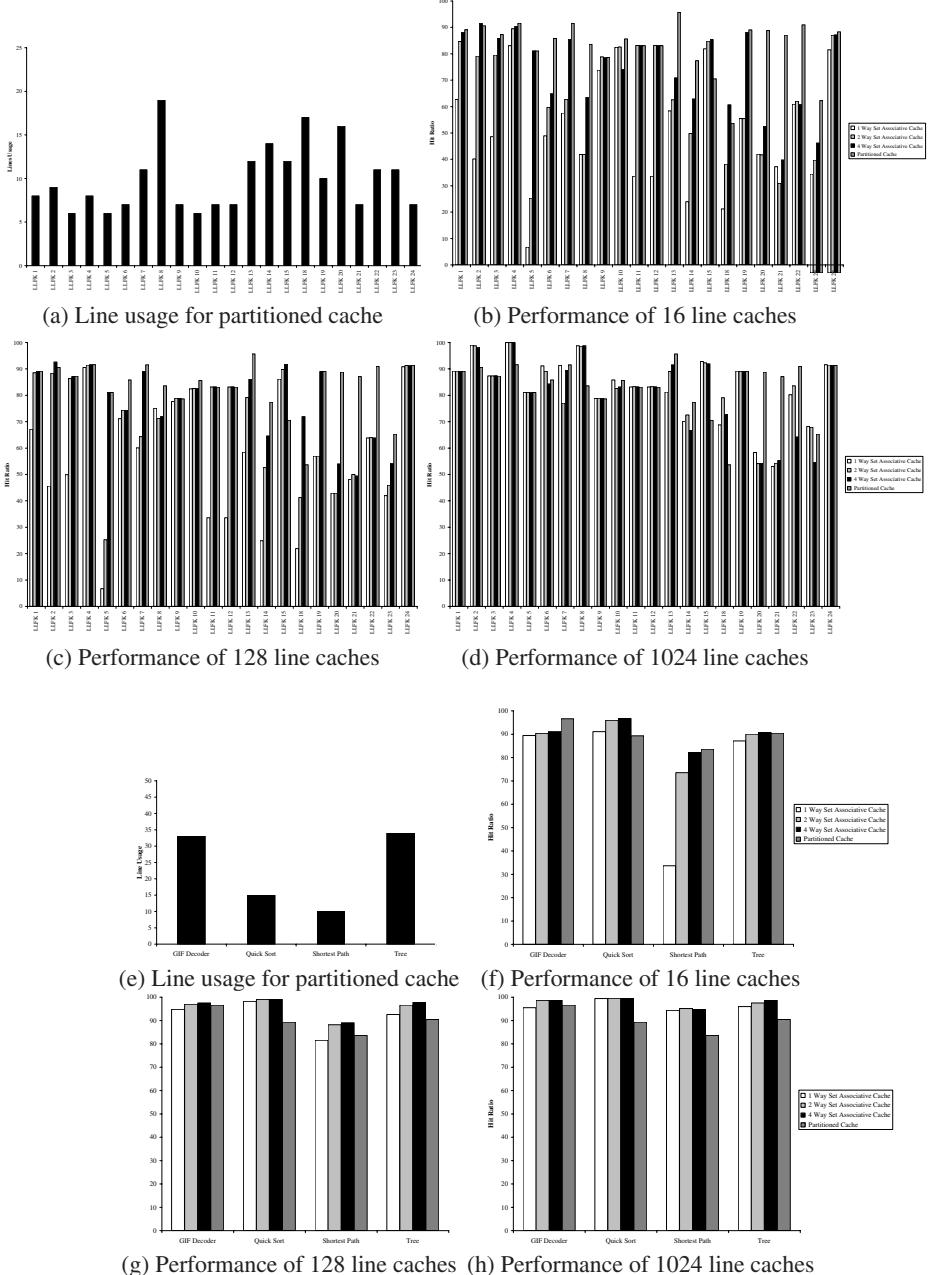


Fig. 2. Performance figures for Livermore (a-d) and Symbolic programs (e-h)

and are now developing a way in which the compiler will be able to select the appropriate one to use in specific cases.

The third property of a collection of references is the *window*. The window of a group defines the range of different data items which are accessed with the same stride. We can see the meaning of a window using the following statement:

```
x = x + y[i*6] + y[i*6+2] ;
```

In this case, *y* has one group with stride 6. The offsets are 2 away from each other ($i*6+2-i*6$), therefore the window is 3.

For one-dimensional variables, the window is computed from the group of references as the distance between the minimum and maximum offset expressions in the dimension used to compute the stride. If the variable has more than one dimension, then the window computation is more complex.

We create one special partition for the scalars (this partition is only used if registers are spilled to memory), one for the stack, and then one partition for every other group.

For all groups of non-scalar references, we create a unique partition. The stride for each partition is the stride of the group times the size of each element in the group. Where the group stride is zero, the partition stride is the size of the elements in the group. The stride is the *mapping function* of the cache partition as described in Section 2.1. The compiler allocates partition sizes large enough to contain the whole window of the group.

3 Results

We have developed a simulator for the partitioned cache scheme, and a compiler to allocate partitions using the algorithms outlined earlier. We have compiled several benchmark applications (including the Lawrence Livermore Fortran Kernels [2] and the MediaBench suite [3]), and executed them on the simulator. All our simulated caches use a write-through allocate policy, optimised write traffic is studied in papers such as [4]. We have compare cost (size of the cache), and performance (miss ratio of the cache) as well as instantaneous hit rate and instantaneous locality [5].

First, we ran the benchmarks on the partitioned cache. This gave us the size and performance figures for our partitioned caches. Then we ran the benchmarks on standard caches (direct mapped, set associative, skewed set associative, victim, and column associative caches).

The figures for the Livermore Kernels and four symbolic programs (a GIF-decoder etc) are presented in Figure 2. The first graph shows the number of lines used by the partitioned cache for each program. The other graphs show performance of some example caches plotted against the performance of the partitioned cache.

The results for the Livermore Kernels show that the partitioned cache achieves a hit ratio equal to or greater than the conventional caches in almost all cases, sometimes outperforming the other caches by a factor of 10. The other benchmarks show similar results [1]. The trend is that partitioned caches deliver performance better than, or comparable with, standard caches and use far fewer cache lines. With a conventional direct mapped cache, all of the lines are used because the range of data addresses is normally much larger than the range of cache addresses. Re-use of cache lines is poor in contrast to the partitioned cache which forces re-use of lines.

4 Related and Future Work

Many standard techniques for caches and buffering can be applied to extend our microcache. For example there has been considerable work on prefetching such as that of Chen and Watson [6,7] which could easily be added to the microcache. Adding prefetching, controlled on a partition by partition basis, gives the partitioned cache the properties of stream-buffers.

We expect that we can improve the compiler considerably using work which has been done on (nested) loop analysis and loop restructuring [8,9]. When the analysis of nested loops is improved, we expect that we will be able to extend our partitioning to other parts of the memory hierarchy, including multi-level caches and virtual memory.

The idea of exposing the cache to the compiler or programmer is not new. It has been proposed before, by, amongst others, Mueller [10], Wagner [11], Kirk [12] and Juan [13]. The microcache system benefits from using a combination of hardware and software to attack the problem. This allows the microcache to deliver features such as fine grain, per object-reference partitioning, which would be costly under a software only system like that proposed by Mueller [10] (that discusses partitioning on a priority basis), and simple hardware logic, that is hard in an unassisted, hardware only system like Wagners [11]. Kirk [12] describes a system that partitions on a per task basis, this offers no inter object interference removal within a task achieved with our proposal. Juan [13] proceed with a technique similar in many ways to the ideas presented here. However, they lack the ability to have multiple partitions per object and do not provide stride parameters in the cache.

Most research involving hardware based partitioning uses a hand coded approach to partition allocation which is laborious and error prone. Our solution is totally automatic.

5 Advantages of Microcaches

Because the partitioned cache uses fewer lines, the physical size of the cache can be much smaller. This reduces the chip size and also increases yield, giving rise to a significant reduction in production cost. Alternatively, the chip area could be used to add additional functions enabling single chip systems to be integrated at low cost for devices such as PDAs and smart cards. Because the physical size of the cache is smaller it will reduce thermal management problems and power consumption, extending battery life in portable systems.

A microcache can use different implementation techniques from those used for traditional caches. Because it is much smaller, it can be implemented using faster, register style, memory. This improvement in the speed of cache access will potentially result in improved performance even when the overall hit ratio of the partitioned cache is lower than a traditional cache.

The protective nature of partitions means the performance of a system is directly related to the performance of its constituent parts. This is important for audio and video applications, where the performance of the individual functions must be maintained. Using ordinary caches or previous partitioned cache systems, the combining of two functions in a program can lead to unpredictable or poorer cache performance.

We believe that microcaches will rekindle interest in multithreaded architectures. A multithreaded processor would need a very large conventional cache to overcome

cache-interfere between threads. Instead, it could be equipped with a moderately sized microcache, which prevent interference by partitioning.

6 Conclusions

We have shown in this paper how to build and program caches that are very small, yet have a high hit ratio. The advantages of small caches are numerous: small caches are faster, they take less chip space, and use less power.

Unlike a conventional cache where data items dynamically compete for cache space, we have given the compiler control of the cache allocation, in the same way that it has control of register allocation. Preventing interference between data object means we can make the cache much smaller while maintaining high performance.

We expect that a microcache will benefit many application areas. We have already shown that on scientific and multimedia programs the use of a microcache can improve price performance of the cache by up to two orders of magnitude. We are also interested in the potential effects on multithreaded architectures, where it should be possible to run multiple threads without interference in a conventional size partitioned cache.

References

1. D. May, D. Page, J. Irwin, and H. L. Muller. Microcaches. Technical Report CSTR-98-010, <http://www.cs.bris.ac.uk/Tools/Reports/Keywords/Predictable.Computing.html>, Department of Computer Science, University of Bristol, October 1998.
2. F. McMahon. *The Livermore Fortran Kernels: A Computer Test Of The Numerical Performance Range*. Lawrence Livermore National Laboratory, Livermore, California, Dec 1986.
3. C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems. In *MICRO-30*, 1997.
4. N. Jouppi. Cache Write Policies And Performance. Technical Report 91/12, Digital Western Research Laboratory, Dec 1991.
5. D. B. Weikle, S. McKee, and W. Wulf. Caches As Filters: A New Approach to Cache Analysis. *Proc. Sixth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 1998.
6. T.-F. Chen and J.-L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. In *iscas94*, pp 223–232, Chicago, Illinois, Apr 1994. ieeecspress.
7. I. Watson and A. Rawsthorne. Decoupled Pre-Fetching for Distributed Shared Memory. In *Proceedings of the 28th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, 1995.
8. M. Wolf and M. Lam. A Data Locality Optimizing Algorithm. *SIGPLAN notices*, 26(6), June 1991.
9. K. S. McKinley and O. Temam. A Quantitative Analysis of Loop Nest Locality. In *asplos7*, Boston, MA, Oct 1996. ACM.
10. F. Mueller. Compiler Support for Software-Based Cache Partitioning. In *ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems*, pp 137–145, Jun 1995.
11. R. A. Wagner. Compiler-Controlled Cache Mapping Rules. Technical Report CS-1995-31, Duke University, Dec 1995.
12. D. B. Kirk. SMART (Strategic Allocation for Real-Time) Cache Design. In *IEEE Symposium on Real-Time Systems*, pp 229–237, Dec 1989.
13. T. Juan, D. Royo, and J. J. Navarro. Dynamic Cache Splitting. *XV International Conference of the Chilean Computational Society*, 1995.

Improving Data Value Prediction Accuracy Using Path Correlation

Wamsi Mohan and Manoj Franklin

Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742, USA
`wamsi@eng.umd.edu` and `manoj@eng.umd.edu`

Abstract. Recent research has shown that it is possible to overcome the dataflow limit by predicting instruction results based on previously produced values or a sequence thereof. Instruction results often depend on the path used to arrive at that instruction; by maintaining different data value histories for the different paths leading up to an instruction, it is possible to do better predictions. This paper studies the effect of control flow correlation schemes on stride-based data value predictors. Our studies indicate that if enough hardware resources can be provided for storing the histories corresponding to different paths, the prediction accuracy increases steadily as the degree of correlation is increased.

1 Introduction

Data Value Prediction (DVP) is becoming an interesting technique to boost instruction-level parallelism (ILP). Predicting an instruction's result allows instructions that depend on that value to execute earlier, thereby allowing concurrent execution of multiple instructions on a multiple-issue processor. DVP was first investigated by Lipasti et al [1], with a last value based predictor. Using stride, two-level, and hybrid predictors, the prediction accuracy can be substantially improved [4].

Several studies in the context of branch prediction have observed that the outcome of a conditional branch instruction is often dependent on the dynamic path leading up to the branch [2] [5]. Similar studies with data value prediction [4] showed that such a correlation exists to some extent when the last outcome based predictor is used. A recent study with stride-based data value predictor also confirmed the positive effect of such correlation on prediction accuracy [3]. To gain better insight into why correlation exists, let us consider a simple case of how the data values can vary depending on the dynamic path. Consider the following C code. Depending on the path taken `var2` evaluates to a different value, and knowing the dynamic path helps to predict `var2` accurately. Similarly, the value of `var3` also depends on the dynamic path.

```
if (cond1) {  
    i++;    k++;    var1 = i;    }  
else {  
    j++;    k = k + 2;    var1 = j;    }  
var2 = k + 10;    var3 = var1 + 5;
```

The studies in [4] [3] applied correlation by considering the outcome of k preceding branches, and storing separate value histories for each of the 2^k possible k -bit patterns. Figure 1 gives a block diagram of such a Degree- k correlation-based value predictor.

A k -bit shift register called Branch History Register (BHR) is used to record the outcome of the last k branches. Each VHT (Value History Table) entry stores 2^k different histories for the instruction currently mapped to it; each history corresponds to a particular k -bit pattern in the BHR. When an instruction address is supplied, the corresponding VHT entry is selected, and the 2^k different histories are read out. A $2^k:1$ MUX is used to select the history corresponding to the k -bit BHR value.

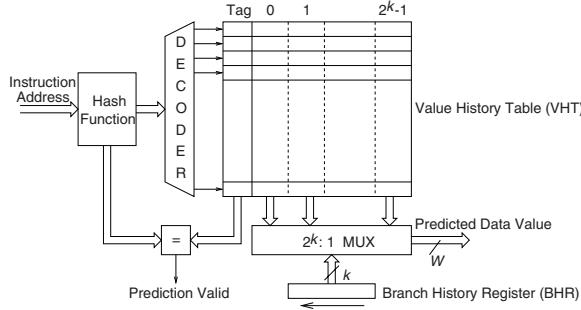


Fig. 1. Block Diagram of a Correlation-based Value Predictor

2 State Machine for Stride Predictor

When control flow correlation is employed in a stride-based predictor, the predictor needs to store a separate stride value for each possible path, because a variable may get incremented/decremented differently in each path. For instance, in the earlier example code, the variable `k` gets incremented by 1 in the `if` path, and by 2 in the `else` path. Thus, to predict `var2` correctly, two separate strides have to be recorded in its history. Once the dynamic path is known, the appropriate stride can be added to the previous (latest) value of `var2` to get the predicted value.

In some situations, it is necessary to store separate “last values” for each possible path. For instance, in the previous example code, in order to predict `var3`, we need to store its last value along the `if` path as well as its last value along the `else` path.

In order to capture both scenario, for each VHT entry, we propose to store both the “global last value” and the “path-specific last value”. The state machine used for each path in a VHT entry is given in Figure 2. Every time a match is detected between the PIR value and a Path Tag value, the path attributes (in the VHT entry) are updated.

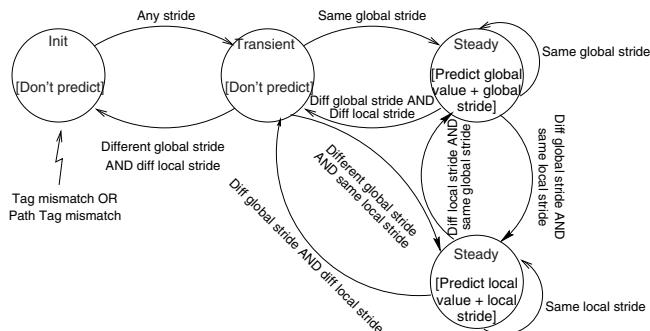


Fig. 2. State Transition Diagram of Stride Predictor

3 Depicting Path Identity Accurately

In the control flow correlation investigated in [4] [3], the path identity is determined by the sequence of recent branch outcomes (the contents of the BHR in Figure 1). In many cases, different dynamic paths may have the same BHR value. One way to overcome this problem is to include information from a path’s instruction addresses when depicting the path (as done for branch prediction in [2]). In particular, the path identity is determined by the sequence of target addresses (taken address or fall-through address) of the conditional branches present in the path. To have a realistic implementation, only some bits of the target addresses are used to determine the path identity.

Figure 3 illustrates the structure of the path identity-based predictor. Apart from the Tag and Global Value fields—which are common to all paths in a VHT entry— p sets of path attributes are also present. Each path attribute has the following fields: Path Tag, State, Stride, and Value. The current path’s identity is stored in the Path Identity Register (PIR); the bits in the PIR act like a FIFO and get shifted out as new target addresses are encountered. Once a VHT entry is selected and the Tag matches, the PIR value is compared against the existing Path Tags. If none of the Path Tags match, then a vacant path attribute field (if any) is allotted to the current path. If there is no vacant path (which can happen when more than p paths exist to an instruction), the LRU (least recently used) path is reclaimed for the current path.

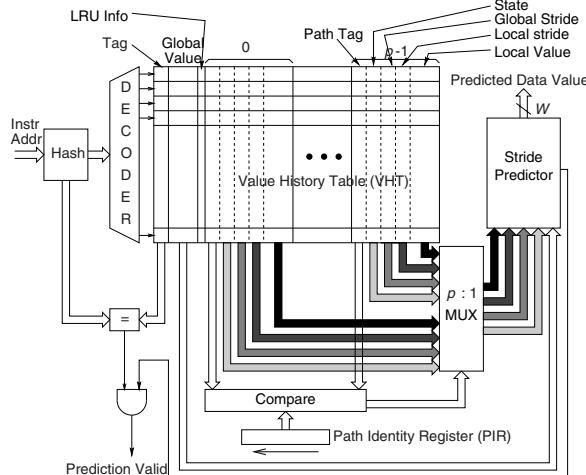


Fig. 3. Block Diagram of a Path Identity Correlation (PIC) based Stride Predictor

4 Simulation Results

The simulation studies are conducted using a MIPS instruction set architecture (ISA) simulator that accepts executable images of MIPS programs. The benchmarks are selected from the SPEC’95 integer benchmark suite. All programs are compiled using the MIPS C compiler with the optimization flags distributed in the SPEC benchmark makefiles. Only the single register producing instructions are considered for DVP. All the results are expressed as a percentage of these value prediction eligible instructions. Each benchmark program is simulated for the first 200 million instructions. For the path identity based correlation scheme, 4 LSBs of the target address is shifted into the PIR each time.

4.1 Results with Varying Correlation Degrees

Table 1 presents the percentage of correct predictions and incorrect predictions obtained for different degrees of correlation, all using a 16K entry VHT with 8 paths per VHT entry. The various columns of numbers in the tables correspond to different degrees of correlation. The first column of numbers refers to degree of correlation zero (this is the same as the regular stride), the second to degree one, and so on. We include both the branch outcome correlation (BOC)-based stride predictor and the path identity correlation (PIC)-based stride predictor in our simulations.

Table 1. Results with 16K entry VHT and 8 Paths per VHT Entry

	Program	Correct Predictions/Incorrect Predictions with Degree of Correlation					
		0	1	2	3	4	5
B	compress	44.7%/2.6%	49.2%/2.2%	50.8%/1.9%	52.1%/1.9%	52.4%/1.9%	52.5%/1.8%
	gcc	48.2%/3.4%	50.4%/3.4%	52.7%/3.5%	54.4%/3.5%	56.3%/3.6%	57.7%/3.7%
	go	39.3%/4.0%	41.6%/4.3%	44.5%/4.3%	46.4%/4.5%	48.3%/4.6%	49.1%/4.7%
	li	43.0%/4.9%	48.1%/3.3%	51.5%/3.4%	53.2%/3.7%	55.5%/3.7%	57.7%/3.9%
P	compress	44.7%/2.6%	48.9%/2.1%	51.0%/1.9%	52.0%/1.9%	52.8%/1.9%	53.3%/1.9%
	gcc	48.2%/3.4%	51.3%/3.4%	54.8%/3.5%	56.8%/3.4%	58.1%/3.3%	58.1%/3.2%
	go	39.3%/4.0%	41.9%/4.3%	44.7%/4.4%	47.0%/4.5%	47.2%/4.4%	46.8%/4.2%
	li	43.0%/4.0%	48.6%/3.5%	53.8%/3.9%	55.9%/3.9%	58.9%/3.8%	59.1%/3.6%

These results indicate that as long as sufficient hardware resources can be allocated (in this case, the number of paths per VHT entry), increasing the degree of correlation results in a monotonic increase in the number of instructions correctly predicted, for both BOC and PIC¹. This means that the time required to initialize the history of each unique path is not affecting the accuracy of prediction (at least up to a correlation degree of 5). This is because of two reasons: (i) The number of times a prediction could not be made due to lack of sufficient history for that specific path is more than offset by the number of times the history is “purer”. (ii) The state machine of the stride predictor does not take too long to start making predictions. It can also be seen that, by taking advantage of control flow correlation, the percentage of mispredictions can sometimes be reduced, especially for `compress`. For the remaining benchmarks (except `go`), the number of mispredictions is minimum when the correlation degree is moderate.

4.2 Results with Fixed Hardware Cost

In the previous experiments, the hardware requirements grow as the correlation degree is increased, because of storing the histories corresponding to different paths. The size of a VHT entry is almost directly proportional to the number of paths in a VHT entry. In the next set of experiments, we fix the product of the number of VHT entries and the number of paths in a VHT entry as 16K. Five different configurations (# VHT entries \times #Paths per VHT entry) are simulated. In particular, given a fixed hardware budget, we are interested in knowing if it is beneficial to have more VHT entries or more paths per VHT entry (with an appropriate increase in correlation degree).

¹ Our experiments with 4 paths per VHT entry indicate that the number of correct predictions tend to decrease for high correlation degrees, because of the conflict between multiple paths.

Table 2 presents the results obtained for both BOC and PIC schemes. For each configuration and benchmark, the correlation degree that gives the highest percentage of correct predictions is reported (this degree is marked within parentheses in the table). We can see that different programs behave differently when the hardware cost is fixed. For `compress` and `li`, the maximum value for the correct predictions occurs when the number of paths is maximum. By contrast, `gcc` and `go` seem to require more number of entries in the VHT; in fact, for `go`, the maximum occurs when the number of VHT entries is maximum. This means that when the hardware cost is fixed, it is better to use VHT mapping schemes that can cater to both cases.

Table 2. Results with Fixed Hardware Cost (# VHT entries \times # Paths per VHT entry = 16K)

	Program	Correct Predictions/Incorrect Predictions (Best Correlation Degree) with # VHT entries \times # Paths per VHT entry =				
		1K \times 16	2K \times 8	4K \times 4	8K \times 2	16K \times 1
B	<code>compress</code>	52.5%/1.8%(5)	52.5%/1.8%(5)	52.3%/1.9%(4)	49.2%/2.2%(1)	44.7%/2.6%(0)
	<code>gcc</code>	42.5%/2.9%(5)	47.6%/3.1%(5)	49.2%/3.3%(3)	48.9%/3.3%(1)	48.2%/3.4%(0)
O	<code>go</code>	32.3%/3.7%(5)	35.2%/3.8%(4)	35.5%/3.7%(2)	38.5%/3.6%(2)	39.3%/4.0%(0)
	<code>li</code>	52.7%/3.4%(5)	53.6%/3.5%(5)	53.1%/3.4%(5)	47.0%/3.1%(2)	43.0%/4.0%(0)
P	<code>compress</code>	53.3%/1.9%(5)	53.3%/1.9%(5)	52.7%/1.8%(4)	48.9%/2.1%(1)	44.7%/2.1%(0)
	<code>gcc</code>	44.2%/2.3%(5)	47.7%/2.6%(5)	49.0%/2.9%(3)	49.2%/3.3%(1)	48.2%/3.4%(0)
I	<code>go</code>	30.4%/2.8%(5)	33.8%/3.4%(3)	34.4%/3.6%(2)	37.8%/3.9%(1)	39.3%/4.0%(0)
	<code>li</code>	58.2%/3.3%(5)	55.3%/3.1%(5)	51.6%/3.1%(4)	46.6%/3.0%(2)	43.0%/4.0%(0)

5 Conclusions

This paper investigated techniques for improving the data value prediction accuracy by incorporating techniques to make use of control flow correlation. In particular, a modified stride predictor was developed to take advantage of correlation. We also studied the feasibility of enhancing the path identity information by considering the target addresses of control flow changing instructions. Our studies indicate that if enough hardware resources can be allotted, then increasing the degree of correlation increases the number of instruction results that can be correctly predicted. On the other hand, when the hardware budget for the predictor is restricted, some programs benefit more from having more entries in the predictor as opposed to having more correlation.

References

1. M. H. Lipasti and J. P. Shen, "Exceeding the Dataflow Limit via Value Prediction," *Proc. 29th Int'l Symposium on Microarchitecture*, pp. 226-237, 1996.
2. R. Nair, "Dynamic Path Based Branch Correlation," *Proc. 28th Int'l symposium on Microarchitecture*, 1995.
3. T. Nakra, R. Gupta, and M. L. Soffa, "Global Context-Based Value Prediction," *Proc. Int'l Symposium on High Performance Computer Architecture*, 1999.
4. K. Wang and M. Franklin, "Highly Accurate Data Value Prediction," *Proc. 3rd Int'l Conference on High Performance Computing*, pp. 358-363, 1997.
5. C. Young and M. Smith, "Improving the Accuracy of Static Branch Prediction using Branch Correlation," *Proc. ASPLOS-VI*, 1994.

Performance Benefits of Exploiting Control Independence

Sreenivas Vadlapatla and Manoj Franklin

Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742, USA
`vsreeni@eng.umd.edu` and `manoj@eng.umd.edu`

Abstract. Many studies have shown that significant levels of parallelism can be extracted from ordinary programs if a processor can accurately look ahead arbitrarily far into the dynamic instruction stream. Control flow changes caused by conditional branches are a major impediment to determining which of the distant instructions belong to the dynamic instruction stream. This paper highlights the importance of exploiting control independence information for extracting this “distant parallelism”. We describe a methodology to find the maximum parallelism available when exploiting control independence. Our study with this tool shows that putting control independence to work has the potential to provide high performance.

1 Introduction

Many studies have shown that significant levels of parallelism can be extracted from ordinary programs if a processor can accurately look ahead arbitrarily far into the dynamic instruction stream—by means of perfect branch prediction and an infinitely large scheduling buffer [4] [5]. Although both of these assumptions are idealistic, the high values of parallelism, nevertheless, give an impetus to explore techniques to extract that parallelism in future processors. Current processors extract parallelism only across a narrow window of the dynamic instruction stream; these systems have no ability to identify distant instructions that can be executed much earlier.

Control flow changes caused by conditional branches are a major impediment to determining which of the distant instructions belong to the dynamic instruction stream. Modern processors typically attempt to overcome this impediment by predicting the outcome of conditional branches, and doing speculative execution along the predicted path. However, in spite of the availability of highly accurate branch prediction schemes, control speculation by itself cannot possibly get to the distant code, because the average distance between two mispredictions is only of the order of 100 instructions or so. Each branch misprediction causes the rest of the speculated path to be abandoned and a different path to be pursued, although the new path may have many instructions in common with the previously abandoned incorrect path. Identifying and exploiting control independences seem to be the natural way to deal with the problem of getting to the distant code [3] [4] [7]. Effective use of control independence information helps to reach distant code, despite the presence of mispredicted branches in between.

The goals of this paper are (i) to present a methodology to accurately model speculative execution of multiple control-independent regions of code, and (ii) to study

the amount of parallelism available when exploiting control independence. A good understanding of the nature of parallelism is necessary before devising good mechanisms to exploit control independence.

2 Methodology for Exploiting Control Independence

Control dependence analysis [2] can determine the control dependences present in a section of code, and processors can use control dependence information to execute control-independent portions of code in parallel, by means of multiple hardware sequencers¹. For instance, when control reaches basic block A in Figure 1(a), such a processor knows that a control-independent section of code starts at basic block C, and it uses two separate hardware sequencers to fetch and execute these two control-independent threads, as shown in Figure 1(b).

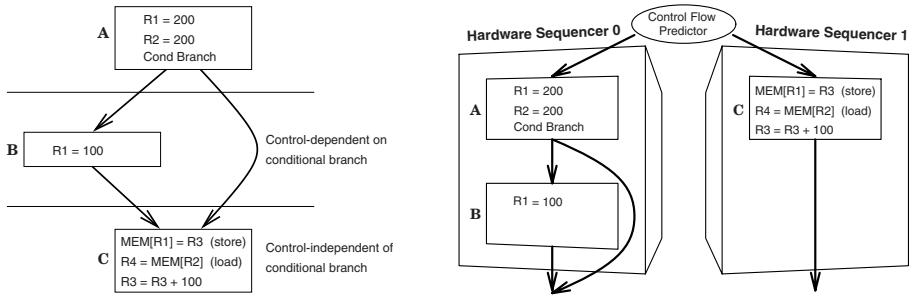


Fig. 1. (a) Example Control Flow Graph (CFG). (b) Executing control-independent threads in parallel with multiple hardware sequencers

The processor performs out-of-order fetch of instructions, and the instruction window, at any given time, may not be a contiguous set of dynamic instructions. Notice that there may be data dependences (through registers and memory locations) from one thread to the subsequent ones. This makes it different from *multiprocessing*, where communication between control-independent threads are engineered by means of explicit synchronization mechanisms.

When pursuing independent flows of control, control speculation needs to be done within each thread [4] [6] [7]; otherwise, the amount of parallelism that can be exploited is limited. That is, when a control sequencer encounters a conditional branch in a thread, it must predict the branch outcome in order to decide one of the control-dependent alternative paths and speculatively execute that control-dependent path,

¹ It is possible to exploit control independence in a limited manner within a single hardware sequencer that can fetch only along a single flow of control at any given time. The instruction window in that case is a contiguous set of dynamic instructions from the speculated path. On detecting a control misprediction, control independence can be exploited in such a processor by selectively deleting one set of (control-dependent) instructions from this window and/or by selectively introducing another set of (control-dependent) instructions into this window.

instead of waiting for the branch to be resolved². When a misprediction is detected in one of the threads, the hardware will selectively squash only the portion of code that was speculatively fetched by that control sequencer (i.e., only the code that is control dependent on the mispredicted branch), and not the subsequent control-independent threads fetched by other control sequencers. Notice, however, that the data dependence effects of the misprediction may spread to subsequent (control-independent) threads. Notice also that multiple branch mispredictions can be handled simultaneously when they are in different threads.

3 Parallelism Measurement

3.1 TAPE: Tool for Available Parallelism Estimation

To measure the parallelism available under various machine models, we developed a tool called *TAPE* (*Tool for Available Parallelism Estimation*), similar to the *Paragraph* tool described in [1] for measuring the parallelism under the single flow of control model. Figure 2 depicts the internals of TAPE. Instruction traces are generated by an instruction set architecture (ISA)-level simulator. The traces are augmented with information about the incorrectly speculated paths. The traces are supplied to the DDG analyzer, which constructs a dynamic dependence graph (DDG) based on the data dependences in the trace, the control flow constraints as per the machine model simulated and the predictions supplied by the predictor, the renaming constraints, and the resource constraints. The DDG is a partially ordered, directed, acyclic graph, in which the nodes represent the computation operations corresponding to the instructions, and the edges represent the run-time dependences between the instructions. Notice that only a single pass is made through the trace and each instruction in the trace is processed only once. The DDG is always kept in topologically sorted form; instructions in the same level of the DDG can be executed in parallel by the abstract machine modeled. The average number of instructions per DDG level gives the *available parallelism* that can be extracted from the program by the abstract machine modeled.

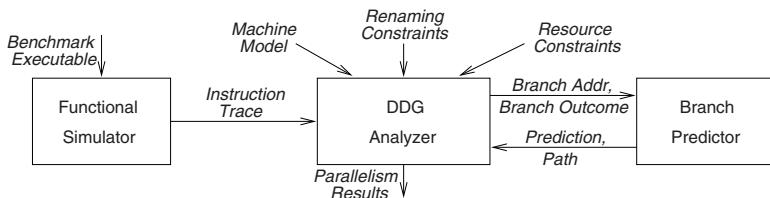


Fig. 2. Block Diagram of TAPE

For conducting the parallelism studies, we used the MIPS-I ISA, and single-cycle functional units. For benchmarks, we used programs from the SPEC '92 and SPEC '95 benchmark suites. The benchmarks are simulated for 500 million instructions. For performing branch predictions, we used a 2-level Pap scheme [8], with a direct-mapped

² It is certainly possible to pursue both outcomes of the branch simultaneously; however, such an approach leads to exponential growth in hardware requirements if many unresolved branches are present in a thread.

16K-entry Branch History Table, a pattern size of 6, and 3-bit saturating counters in the Pattern History Table entries. The second column in Table 1 gives the branch prediction accuracies that we obtained in our studies. Although the exact parallelism values presented in this section are dependent on the sophistication of the compiler, and can potentially vary if a different compiler or ISA is used, we believe that these results represent the general trends.

Benchmark	Branch prediction accuracy	Available parallelism with		
		No control independence	Control independence	Perfect branch prediction
compress95	88.88%	10.5	162	563
espresso	95.90%	13.5	120	1617
go	87.80%	8.5	51	104
m88ksim	98.52%	14.8	397	615
vortex	98.59%	38.0	1289	1771
dnasa7	99.49%	26.5	1424	1878
fpppp	98.45%	740.0	742	745

Table 1. Branch Prediction Accuracies and Available Parallelism

3.2 Available Parallelism

Table 1 presents the available parallelism obtained for three abstract machine models (given in three columns). Notice that the parallelism obtained with the nCI model—ranging from 8.5 (for go) to 740 (for fpppp)—is higher than those reported in previous studies [4]. The main reasons for this improvement are: (i) the improvement in branch prediction accuracies that we obtain (cf. Table 1) due to the better branch prediction schemes available today, and (ii) the parallel execution of branch instructions by today’s nCI processors.

As expected, the parallelism obtained with control independence is significantly higher than that without control independence for all benchmarks. This is to be expected for benchmarks like go, which have poor branch prediction accuracies. Even for benchmarks like m88ksim, vortex, and dnasa7, which have very high branch prediction accuracies (98.52%, 98.59%, and 99.49%, respectively), utilizing control independence helps to get significantly higher levels of parallelism. The main reason for this is that many mutually data-independent sections of code in m88ksim, vortex, and dnasa7 lie far apart in the dynamic instruction stream. Even if only a single branch is mispredicted between these data-independent code sections, it acts as a barrier when control independence is not exploited, preventing the parallel execution of these code sections.

4 Discussion and Conclusions

Today’s state-of-the-art processors perform aggressive speculative execution along a single flow of control. A limitation of this approach is that each branch misprediction causes all of the subsequent instructions in the instruction window (from the speculated path) to be discarded, without retaining any control-independent instructions that may be present in the instruction window. Recognizing control-independent sections of code, and performing speculative execution along multiple independent flows of control has the potential to extract the large amounts of parallelism available at a distance.

We have already begun to see descriptions of execution models and implementations that attempt to exploit control independence in a limited manner. All of these use multiple hardware sequencers to fetch control-independent portions of code, and execute them in parallel. In these processors, instructions from far ahead in the instruction stream can be fetched, decoded, and executed even before the up-stream instructions have been fetched. Of course, they rely on some means to determine the data dependences between the not-yet-fetched, up-stream instructions, and the currently executed down-stream instructions. Also, when a misprediction is detected in an up-stream flow of control, which causes a different flow of control to be pursued by the up-stream sequencer, the updated data dependence information is conveyed to the down-stream sequencer.

Our experimental results with infinite size instruction windows show that the maximum parallelism available when exploiting control independence is significantly higher than that available when control independence is not exploited. It remains to be seen what ingenious schemes computer architects would devise in the future to overcome the implementation hurdles of independent flows of control, so as to extract the large amounts of parallelism available at a distance.

Acknowledgements

We wish to acknowledge the financial support of the National Science Foundation through the grant CCR-9711566.

References

1. T. M. Austin and G. S. Sohi, "Dynamic Dependency Analysis of Ordinary Programs," *Proc. 19th Int'l Symposium on Computer Architecture*, pp. 342-351, 1992.
2. R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and F. Zadeck, "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph," *ACM Trans. Programming Languages and Systems*, Vol. 13, No. 4, pp. 451-490, Oct 1991.
3. P. Dubey, K. O'Brien, K. M. O'Brien, and C. Barton, "Single-Program Speculative Multithreading (SPSM) Architecture: Compiler-assisted Fine-Grained Multithreading," *Proc. Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT '95)*, 1995.
4. M. S. Lam and R. P. Wilson, "Limits of Control Flow on Parallelism," *Proc. 19th Int'l Symposium on Computer Architecture*, pp. 46-57, 1992.
5. M. Postiff, D. Greene, G. Tyson, and T. Mudge, "The limits of instructions level parallelism in SPEC95 applications," *Proc. 3rd Workshop on Interaction Between Compilers and Computer Architecture (INTERACT-3)*, 1998.
6. E. Rotenberg, Q. Jacobson, and J. E. Smith, "A Study of Control Independence in Superscalar Processors," *Proc. Int'l Symposium on High Performance Computer Architecture (HPCA)*, 1999.
7. K. B. Theobald, G. R. Gao, and L. J. Hendren, "On the Limits of Program Parallelism and its Smoothability," *Proc. 25th Int'l Symposium on Microarchitecture (MICRO-25)*, pp. 10-19, 1992.
8. T-Y. Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proc. 19th Int'l Symposium on Computer Architecture*, pp. 124-134, 1992.

Fast Slicing of Concurrent Programs

Diganta Goswami

Department of Computer Science and Engineering, IIT, Kharagpur, INDIA - 721 302
email: diganta@cse.iitkgp.ernet.in

Rajib Mall¹

Department of Computer Science, Curtin University of Technology
GPO Box U1987, PERTH, Western Australia 6845
email: rajib@cs.curtin.edu.au

Abstract

Debugging complex concurrent programs is a very difficult problem. Slicing techniques can simplify debugging and understanding these programs. However, slicing of concurrent program is compute-intensive as the required intermediate graph often becomes large. In this context, we generalize the structure of the intermediate graph representation of concurrent programs and this leads us to introduce the notion of Concurrent Control Flow Graph (CCFG). We introduce a parallel algorithm to compute static slices of concurrent programs using CCFG.

1 Introduction

A slice extracts only those statements from a program which are relevant to the computation of some given value. Slicing helps in debugging, testing, and maintenance of programs [1]. An excellent survey on program slicing is available in [1]. Many of the present day software systems are actually concurrent or distributed systems. However, it is found in practice that it is much more difficult to debug and understand the behavior of concurrent programs than it is for the sequential counterpart. Non-deterministic nature of concurrent programs, lack of global states, and possible interactions among processes are some of the reasons for this. Therefore, there is a pressing necessity for devising techniques for slicing concurrent programs.

Several methods for computing slices of programs have been reported [2],[3]. The seminal work of Weiser computed the slice of a program from its control flow graph (CFG) representation [2]. A *parallel slicing algorithm* to compute slice for sequential programs was presented by Danicic et al [4]. In their method, a *process network* is constructed from the program to be sliced. The CFG of the program is first constructed to construct the process network. The Reverse CFG (RCFG) is then constructed by reversing the direction of every edge in the CFG. Every node in the RCFC represents a process and the arcs correspond to communication channels. Arcs entering a node i represent inputs to process i and arcs leaving node i represent outputs from process i . Each process sends and receives messages that

¹On leave from IIT Kharagpur

are sets of variables and node identifiers. If the input S to process i contains any of the variables defined by i or contains any of the node controlled by i (in case i is a predicate node), then the process i outputs a message on all its output channels consisting of:

- all its input variables (elements of S) that it does not define,
- all variables that it references,
- its node identifier, i .

Otherwise, the process i merely outputs S on all outgoing channels. To compute a slice for the slicing criterion $< s, V >$, where V is the set of variables, network communication is initiated by outputting the message V from process s . The algorithm computes the slice of a program by including the set of nodes whose identifiers are input to the *entry* node of the CFG.

2 Representation of Concurrent Programs

In our subsequent discussions, we will assume the Unix primitives for process creation, interprocess communication, and synchronization [5]. For computation of static slices, we assume that processes are statically created. The synchronization semantics that we have considered here are of two types – *partially blocking* and *fully blocking*. We represent a concurrent program in three hierarchical levels as described below.

Process Graph: A *process graph* is a directed graph $G_p = (V_p, E_p, s_p)$ with vertex set V_p and edge set E_p . Vertices represent processes. We call the vertices as *process nodes* and denote as $n(p)$. The edges of a process graph represent fork/join relationships among processes. Thus the edges are of two types: *fork edge* and *join edge*. A process node, $n(p)$ is created when there is a fork call in the program. A *fork edge* from a vertex $n(p)_i$ to another vertex $n(p)_j$ exists, if there is fork call in $n(p)_i$ creating $n(p)_j$. A *join edge* from process $n(p)_i$ to $n(p)_j$ indicates that $n(p)_j$ is waiting for $n(p)_i$ to terminate. That is, a join edge represents the fact that terminating processes join to create a new process. We also assume a special *entry* node, s_p representing the beginning of the program. No edge is directed to s_p . There is a special edge $(s_p, n(p)_0) \in E_p$, where $n(p)_0$ represents the beginning of the program.

Concurrency Graph: In a concurrency graph, the process nodes of the process graph containing message passing statements are split up into three different kinds of nodes, namely *send node*, *receive node*, and *statement node*. The code belonging to the process nodes of the process graph are examined to achieve node splitting. Different cases may arise depending upon the presence of `msgsnd()`/`msgrecv()` statements in the nodes in different orders. A send node contains a sequence of statements ending with a `msgsnd()` statement. A receive node contains a sequence of statements beginning with a `msgrecv()` statement. And a statement node consists of a sequence of statements without any message passing statement. Different nodes thus created are connected by appropriate *control edges* to show the control flow. An edge called *synchronization edge* from every send node to the corresponding receive node is constructed. Synchronization edge depicts the fact that the node pointed to by this edge must wait until the statements represented by the node at the other direction completes execution. In case of fully blocking semantics for message passing, the synchronization edge is bi-directional, indicating that both send and receive nodes are synchronization dependent on each other. Each node of the concurrency graph is called a *component* and is denoted as $n(c)$.

Concurrent Components: A *component* is the basic unit of concurrent execution. The maximal set of components which are capable of concurrent execution is called a *concurrent set of components* or just a *concurrent component*. Determination of concurrent components is necessary to handle slicing of concurrent

programs in presence of shared variables. We present here an algorithm for determining the set of components concurrent to a given component. PC_i represents the set of nodes of the concurrency graph which are concurrent to the node $n(c)_i$. And PC is the set of all concurrent components of the concurrency graph. The complexity of the algorithm to compute all concurrent components is found to be $O(N_c \times E_c)$, where N_c is the number of components and E_c is the number of edges in the concurrency graph.

Algorithm for determination of concurrent components

Input: Concurrency graph, (V_c, E_c) .

Output: $PC = \{PC_i | i = 1, 2, 3, \dots |V_c|\}$.

$$PC_i = \{x | x \in V_c \text{ and } x \text{ is concurrent to node } n(c)_i\}$$

For every node $n(c) \in V_c$ **do**

Begin

- Construct the set, $PU = \{j | j \in V_c \text{ and node } n(c) \text{ is reachable from node } j \text{ traversing transitively along fork/control/synchronization edges}\}$
- Construct the set, $PD = \{k | k \in V_c \text{ and } k \text{ is reachable from node } n(c) \text{ traversing transitively along fork/control/synchronization edges}\}$.
- Construct $PC_n = V_c - (PU \cup PD \cup \{n(c)\})$

End

Concurrent Control Flow Graph (CCFG): Concurrency graph shows the dependencies among processes arising out of communication among them via message passing. However, processes may also interact through use of shared variables. Also, to compute a slice, in addition to concurrency and interprocess communication we must also represent the traditional instruction sequences in the program. To achieve this, we construct the next level graph, CCFG. To construct the CCFG we have to first resolve the cases arising due to the use of shared variables.

Shared Dependency: The fork() function call creates a new process called child which is an exact copy of the parent process. The parent and child processes have different copies of all variables [5]. However, a single copy of shared data segment acquired by the shmat() and shmat() function calls is shared by both processes. We handle such accesses to unsynchronized shared variables through construction of a *shared dependency edge*. A shared dependency edge can exist between concurrent components only. After determining the concurrent components, we can find the shared dependency by associating the use of shared variable in one component to the definitions of the shared variables in the other components.

Semaphore Dependency: In the Unix environment, semaphores can be obtained through the semget() call. The value of a semaphore can be set by semctl() call. The increment and decrement operations on semaphores are carried out by semop() call [5]. If there are multiple accesses to shared variables within a (P,V) block, the access is serialized. In case of shared variable access by two concurrent components within a (P,V) block, if one component uses it within a (P,V) block before defining it, then the value of the shared variable in this component will depend on the *last definition* within (P,V) block in the other component. We construct a *semaphore dependency edge* from the statement having last definition within (P,V) block in one component to the statement having first use within (P,V) block in the other component if that component does not define the variable before using it.

Construction of CCFG: To construct the CCFG of a concurrent program we start with the process graph. We first construct the CFG for every process node, $n(p)$ in the process graph, where every statement/predicate is represented by a node and is denoted as $n(s)$. A special node called the *start* node, $n(s)_s$ is introduced to mark the beginning of the CFG for a process node. The node in the CFG of a process node $n(p)$ representing the last statement of $n(p)$ is denoted as $n(s)_l$. For every fork

edge $(n(p)_i, n(p)_{i+1})$ in the process graph, a fork edge in the CCFG is constructed from $n(s)_l$ of $n(p)_i$ to $n(s)_s$ of $n(p)_{i+1}$. This edge represents the possible control flow from the last statement of $n(p)_i$ to the first statement/predicate of $n(p)_{i+1}$. For every CFG of an $n(p)$ we construct a special control flow edge from $n(s)_s$ to the node $n(s)_0$ representing the first statement of $n(p)$. If a node $n(s)_i$ in CFG for $n(p)_k$ represents a `msgsnd()` statement, then a synchronization edge is constructed from $n(s)_i$ to the node $n(s)_j$ in the CFG for some $n(p)_m$, where this node $n(s)_j$ in $n(p)_m$ represents the corresponding `msgrecv()` statement. A shared dependency edge is constructed from $n(s)_j$ to $n(s)_i$ provided

- a shared variable $sh \in ref(n(s)_i)$, and $sh \in def(n(s)_j)$
- $n(s)_i \in n(c)_m$ and $n(s)_j \in n(c)_n$ in the concurrency graph and the components $n(c)_m, n(c)_n$ are concurrent.

A semaphore dependency edge is constructed from $n(s)_j$ to $n(s)_i$ if

- a shared variable $sh \in ref(n(s)_i)$, and $sh \in def(n(s)_j)$,
- $n(s)_i \in n(c)_m$ and $n(s)_j \in n(c)_n$ in the concurrency graph and the components $n(c)_m, n(c)_n$ are concurrent,
- $n(s)_i$ and $n(s)_j$ are within P-V blocks using the same semaphore, and
- definition of shared variable sh in $n(s)_j$ is the last in that P-V block.

3 Parallel Algorithm For Static Slicing

To compute a slice of concurrent programs, we extend the parallel algorithm proposed in [4] to our representation of concurrent programs. If process networks for our proposed representation of concurrent programs are constructed, we will have some additional processes and channels which do not exist in the process networks for sequential programs. We now outline how these processes and channels are handled.

msgsnd(): If a process representing a `msgsnd()` statement receives a message, it simply transmits the message on all output channels. Before sending out the message, it adds its node identifier to the message if it receives the message from a message passing channel, otherwise the message is transmitted unaltered.

msgrecv(): If a process representing a `msgrecv()` statement receives a message, it checks whether the message contains the data that the `msgrecv()` statement receives from the corresponding `msgsnd()` statement. If not, then it transmits the input message on all output channels without any change except the message passing channel. Otherwise, it transmits the input message along with its node identifier on all output channels except the message passing channel, and transmits a new message, $M = \{data\}$, on the message passing channel, where 'data' is the information received by the `msgrecv()` statement from the corresponding `msgsnd()` statement.

fork call: A process representing a fork call always outputs the input message along with its node identifier on all output channels.

Shared/Semaphore Dependency Channel: Whenever a process which is having shared dependency channel as one of its output channels receives a message from its input channel, it checks whether the message contains the relevant shared variable for that process. The behavior of the process is same for all output channels except the shared dependency channel as described in Section 1. But it outputs a new message on the shared dependency channel containing only the shared variable if the input message contains the relevant shared variable for this process, otherwise it does not output any message on the shared dependency channel.

The processes representing the dummy nodes like s_p and $n(s)_l$ simply receive messages from input channels and transmit unaltered through the output channels. The messages sent along shared dependency channels (or synchronization channels),

if any, need to be output only once. This is because of the fact that the message that is output on any one of these channels, is always fixed. For computation of slice we follow the following steps:

- Construct the hierarchical CCFG for the concurrent program,
- Reverse the CCFG,
- Compile the RCCFG into a process network,
- Initiate network communication by outputting the message $\{s, v\}$ from the process representing the statement s in the process P of the reverse CCFG, where $< P, s, v >$ is the slicing criterion.
- Continue the process of message generation until messages generated by each of the processes are found to be new,
- Add to the slice all those statements whose node identifiers have reached the *entry* node of the CCFG.

The termination of the process network for concurrent programs can easily be established. It is proved in [4] that the slices computed by the parallel algorithm for a sequential program is exactly the same as produced by Weiser's algorithm. The same can also be proved for concurrent programs using the same method. Speed up can be achieved when the parallel algorithm is executed on a set of processors. We assume that the basic unit of task that can be assigned to individual processor is all the processes of the process network (representing the nodes of the CCFG) belonging to a process node in the process graph. If N is the number of process nodes in the process graph and $M \leq N$ is the number of processors, then we can achieve at most N/M time speed up over that when running on a single processor.

4 Conclusion

We have presented the extended version of parallel algorithm for static slicing of concurrent programs. The concept of control flow graph has been extended to represent concurrent programs in a hierarchical fashion. When executed on a set of processors, we will get several times speed up as already stated. Our initial experimentation with several example applications has yielded encouraging results.

References

1. Tip, F. : A Survey of Program Slicing Techniques, Journal of Programming Languages, (Sept. 1995), 121-189
2. Weiser, M.: Program Slicing, IEEE Trans. on Software Engg., **10(4)**, 352-357
3. Ottenstein, K., Ottenstein, L.: The Program Dependence Graph in Software Development Environment, Proc. of the ACM SIGSOFT/SIGPLAN Softw. Engg. Symp. on Practical Software Development Environments, SIGPLAN Not., (1984), **19(5)** 177-184
4. Danicic, S., Harman, M., Sivagurunathan, Y.: A Parallel Algorithm for Static Program Slicing, Tech. Report, School of Computing, Univ. of North London, (1995)
5. Bach, M. J. : The Design Of The Unix Operating System, PHI Ltd., New Delhi, (1986)

Session I-B

Cluster Computing
Chair: R. Govindarajan
Indian Institute of Science

VME Bus-Based Memory Channel Architecture for High Performance Computing

Manuj Sharma, Anupam Mandal, B. Shankar Rao, and G. Athithan

Advanced Numerical Research and Analysis Group
Defence Research and Development Organisation
Kanchanbagh, Hyderabad 500 058, India
anurag@hd1.vsnl.net.in

Abstract. The suitability of the multi-master VME bus for the design of a memory channel architecture for high performance computing is discussed. A moderately scalable cluster-based parallel computing system and its communication network based on this architecture are described next. The development of a thirty two processor parallel computing system and its programming environment are outlined. The details of the intra-cluster and inter-cluster communication channels in this system and the interface for using them are presented. Some parallel applications in computational fluid dynamics and the speed-ups achieved are presented next. A discussion on the experiences of building this system and further efforts underway to enhance its scalability conclude the paper.

1 Introduction

Parallel processing has proven to be a cost-effective route to the development of high performance computers[1,2]. Many commercial supercomputers, such as the IBM's SP2, the Cray's T3D and the Intel's Paragon to name a few[3], are built in recent times using the concept of parallel processing. Most of these systems implement a communication network connecting a variable number of processors and provide a message passing library for the development of parallel programs. Notwithstanding the commercial availability of these computers, many groups are trying to exploit the recent high speed networks based on Gigabit Ethernet and ATM switches for configuring networks of workstations as parallel processing systems[4,5]. These efforts are undertaken mostly for economical reasons and also to benefit from the free-time available on workstations present in many organisations.

Parallel processing systems based on switches have the advantage of scalability. However a major drawback of the switch-based communication networks is the high latencies involved in message transfers. Typically the latency could be hundreds of microseconds to even milliseconds, though there are exceptions such as the 60 microseconds latency of the custom-made switch of IBM's SP2[6]. In an effort to reduce the latency down to a few tens of microseconds, some years ago Digital

Equipment Corporation (DEC) proposed a new method of message passing based on what it calls the Memory Channel Architecture[7]. The central idea of the memory channel architecture is to map the pages of physical memory of a processor that is to receive a message onto the virtual address space of the processor that needs to send the message. Such a mapping enables the sender to write the message directly onto the physical memory of the receiver without any intervention from the operating system. Direct writing onto the receiver's memory is expected to involve low communication overheads and hence lead to a low latency. The implementation of the memory channel architecture by DEC, in fact, has validated this expectation.

In view of the distinct simplicity of the concept and also of the merit of low latency, at the Advanced Numerical Research and Analysis Group (ANURAG) at Hyderabad, India we began to develop a series of parallel computers called PACE+ based on the concept of memory channel architecture. The PACE+ systems are aimed at users working in computational fluid dynamics[8]. Applications in fluid dynamics are some of the most compute-intensive ones and demand reliable computing platforms. For the dual purpose of implementing memory channel architecture and developing a reliable parallel computer we chose the VME bus and VME compatible processor boards. The VME bus allows multiple processors to co-exist on a single backplane. These processors can address each other's memory as though their memory is shared, which paves the way for the setting up of memory channels for communications. The details of several configurations of PACE+ designed and developed at ANURAG and the experiences gained in using them are presented in this paper.

The paper is organised as follows. In section 2, the memory channel architecture for parallel computing is discussed. In section 3 a design of memory channel architecture based on the VME bus is presented. The details of the design of PACE+ are also presented in the same section. Sections 4 and 5 describe the communication model and the programming environment in PACE+ respectively. The speed-ups of some applications run on eight and thirty two node PACE+ systems are presented in section 6. Discussions and future plans conclude the paper.

2 Memory Channel Architecture

The overheads in any standard network for message passing limit the ability of a connected cluster of processors to achieve its full potential as a parallel system. Though the network bandwidths are increasing, the latency and the communication overheads associated with a message transfer remain very high, sometimes reaching a value of 1000 times of that encountered in shared memory systems[7]. The source of high latency in standard networks is the operating system overhead associated with communication. Typically, a node initiates a communication by calling a service of the operating system. This service routine that supports the communication is usually complex and consists of three primary layers namely, the application interface, network communication protocol, and the network device driver software. Executing such a protocol stack consumes significant amount of the CPU time, resulting in high latencies.

In a network based on the memory channel concept, the receiving processor's physical memory is mapped onto the sender's virtual address space. This mapping virtually stretches the write-port of the receiving processor's physical memory and makes it available to the sender for write-only purposes[7]. When the sender needs to communicate, the message is written in the suitably mapped physical memory of the receiver without any involvement of the operating system. The result is a much reduced communication overhead and latency.

3 VME Bus-Based Memory Channel Architecture for PACE+

Many applications in defence research and development involve the use of computational fluid dynamics and hence need high speed computing resources. The design and development of combat aircraft is an important one among them. In order to support users of computational fluid dynamics in our organisation, we decided to develop a high speed parallel computer. This system, called PACE+ for 'Processor for Aerodynamic Computation and Evaluation', was to be a scalable parallel computer to be built from commercially available subsystems. Having examined the memory channel concept and its merits, we decided to model the communication network for PACE+ on the memory channel architecture.

The PACE+ system is a cluster-based parallel computer built using the HyperSPARC processors[9]. In order to build the interconnection network based on the memory channel concept, several standard and commercially available bus-backplanes such as the VME, PCI and Multibus etc., were examined. All these buses allow many processors to be connected to a single backplane. Some of them support memory mapping while others support I/O mapping for inter-processor exchanges. Using the PCI bus, as tried by DEC [7], requires the development of custom hardware to perform memory mapping as the PCI bus addresses are I/O mapped. The Multibus-II and the Futurebus are not as widely used nor are they available now from many suppliers as is the VME bus. Keeping its reliability, openness and industry-standard compliance in view, the VME bus was chosen for building the interconnection network of PACE+.

A standard VME backplane comes with twenty one slots and can accommodate eight processor boards comfortably. To connect two or more backplanes one uses a specific hardware called reflective memory board[10]. A pair of these boards can form a memory mapped link between two backplanes. Called a cluster, a backplane with eight processors is the basic unit of PACE+ architecture. Large systems are then built by interconnecting two or more clusters by means of reflective memory links in accordance with any desirable topology. As a design principle, storing and forwarding of messages is not supported in PACE+ systems so as to keep the development simple. Therefore, any two clusters that need to communicate have to be directly connected by reflective memory links.

For the sake of compactness, the processors are connected together using VME backplanes and reflective memory links as diskless single board computers. One of the clusters contains an additional ninth processor called the Front-End Processor (FEP).

The FEP has all peripheral devices, hard disk etc. and is a standard workstation running the Solaris operating system. The FEP provides the interface between the users of PACE+ and the back-end network of processors. The back-end processors, called nodes, run a customised kernel, called the ANUPAM kernel which performs some basic functions such as memory mapping, text pages protection, hardware initialisation, initialisation and clean-up of communication segment and exception handling. Multitasking and I/O operations are not supported on the nodes.

The inter- and intra-cluster communications are performed over the network of VME backplanes and reflective memory links after setting up the required memory channels. To configure the memory channels among others, the local memory in each node is organised as shown in figure (1). One MB at the lower-end of the physical memory is reserved for the page table, interrupt vector table and the data structures and routines specific to the ANUPAM kernel. The user code is loaded starting from the end of this segment and it grows towards the higher-end of the memory. A segment of 4 MB at the higher-end is reserved for communication buffers. A heap starts after the user code and grows towards the communication buffers while a stack grows from the end of the communication buffers towards the heap.

Each reflective memory board has 4MB of memory that is shared between two clusters. In a link consisting of two reflective memory boards, whatever is written on one board's memory gets 'reflected' in that of the other at the rate of the VME standard. The upper half of this 'shared' memory is used by one cluster, say A, to send messages to the other cluster, say B. The cluster B would use the lower half to send messages to cluster A. A brief description of the method of setting up of the memory channels is provided below.

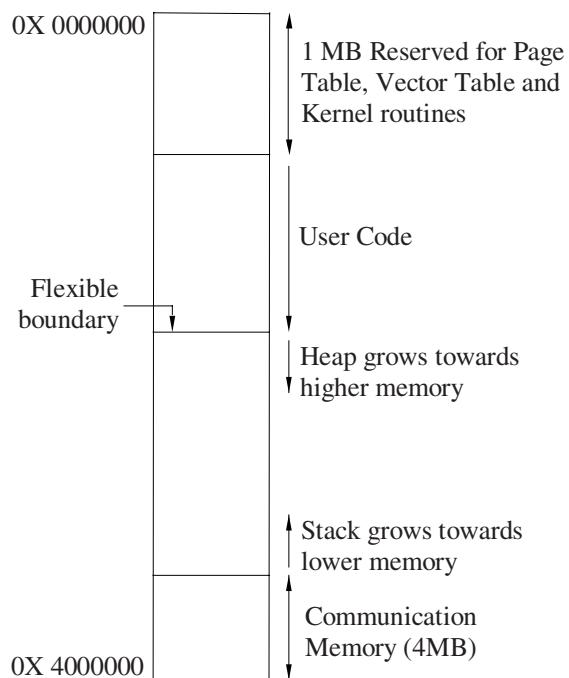


Fig. 1. The organization of local memory at a node

For intra-cluster channels within each cluster, connections for each node in the cluster are created by mapping some specific pages of the node's communication memory onto the corresponding pages in the virtual VME memory of every other node in that cluster. A schematic illustration of this mapping is given in figure (2). Each node in a cluster has seven immediate neighbours in the same cluster from which it can receive messages. Therefore, a segment of each node's communication memory is logically partitioned into seven slots of 512 kbytes each, one slot corresponding to each immediate neighbour. A node that wants to send a message to another writes the message in the slot available for it in the latter's

communication memory. The receiving node needs only to read its local memory for any message that is expected. Thus, a blocking receive would not interfere with the VME bus traffic.

The inter-cluster channels are set up using the reflective memory links. Each node in a cluster maps the physical reflective memory attached to it, into its own virtual address space. The reflective memory board on one cluster is connected to that on the other cluster by a flat cable. The nodes in the other cluster similarly map the local reflective memory onto their virtual address space. Thus, a set of memory channels is established from nodes in one cluster, through the pair of reflective memory boards, to the nodes in the other cluster. Any node in a cluster can write messages into the 'local' reflective memory over the VME backplane, which get transferred to the reflective memory at the other end of the connecting cable so that any node on the other cluster can read its 'local' reflective memory for receiving the message. Figure (2) shows how the reflective memories of two connected clusters are mapped onto the virtual address space of node 0 in each cluster. Between any two connected clusters there are about 64 memory channels supported by 4MB of memory in the associated reflective memory link.

With the implementation of the memory channel network as described above, building an eight or sixteen node parallel computer is rather straightforward. The

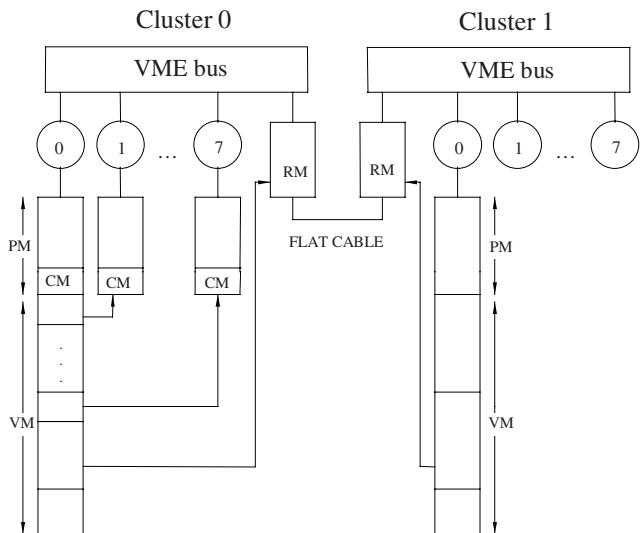


Fig. 2. The mapping of communication buffers of node 1 to node 7 and reflective memory into the virtual address space of node 0. The abbreviations, PM, RM, VM and CM stand for, Physical, Reflective, Virtual and Communication Memories

former is just a single cluster while the latter is two clusters connected by a reflective memory link. In order to support the aerodynamic computational requirements in our organisation, a thirty two node PACE+ was configured using the memory channel network. Comprising four clusters, this system has a reflective memory link between every pair of clusters. Figure (3) shows the details of the interconnection scheme. The FEP is connected to one of the four clusters. Each cluster has three reflective memories and their counterparts sit on the other three clusters, one in each cluster. For the purpose of communications between the FEP and the nodes, additional memory channels are set up whose *send* as well as *receive* buffers are located in the respective cluster's reflective memory.

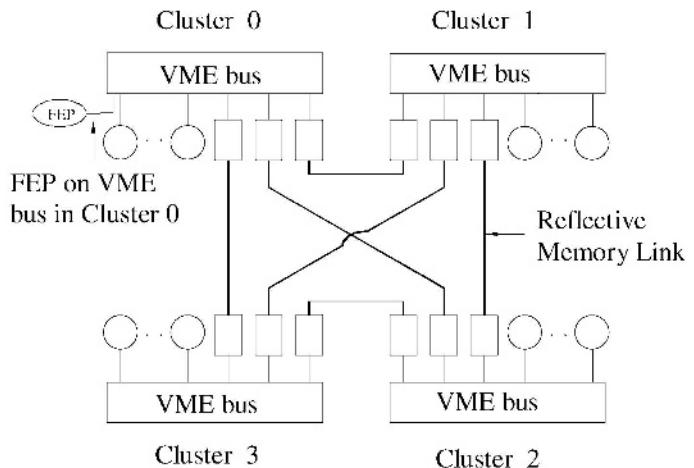


Fig. 3. A block diagram of PACE+ architecture. Within a cluster, a circle denotes a processor and a rectangle denotes a reflective memory board

4 Communication Model

With the provisions of substantial amounts of memory for the inter- and intra-cluster channels, the communications are basically buffered. The sending of messages is non-blocking as long as there is sufficient space in the receiver's buffer. If the size of the message exceeds the available space, then the sender fills the buffer and waits i.e., blocks, till the buffer is cleared by the receiver. The receiving operation is blocking in PACE+ systems.

The buffer is organised in the form of a ring with two associated pointers, namely Read-Head and Write-Head. If the buffer is empty, both the pointers coincide. The sender, after putting the message in the buffer, increments the Write-Head by the size of the message. The receiver reads the message if the pointers do not coincide. After reading, the receiver increments the Read-Head by the size of the message read. This simple method precludes out-of-order message passing between any sender and receiver.

The user interface with the communication network supports basically point-to-point communications. Since each node runs only one process at any time, the source and destination identifiers of the communicating processes are identified with the corresponding logical node numbers in the parallel system. Three kinds of communications are supported in the PACE+ communication model[11]. The first kind enables a user to send or receive a single variable of any basic type. The second kind enables a user to send or receive an array of variables of any basic type. To send a list of variables of different basic types, the third kind of communication is to be used. Besides these, the user is provided with a function that returns the number of nodes in the parallel system. Another function in the user-interface returns the logical process identifier of the process invoking the call. For the purpose of debugging, a special function is provided at the nodes to print messages on terminals connected to the FEP.

Using this interface for the communication network, the latency and the throughput of the network were measured on the intra-cluster channels. Assuming a linear communication overhead model as a function of latency and throughput[12], the transfer rates for three classes of messages were obtained. For the first class, consisting of short messages in the range of 4 bytes to 4k bytes, the latency turned out to be 17.2 microseconds justifying the usage of memory channel approach for building the communication network. For the second class, consisting of medium size messages in the range of 4k bytes to 32k bytes, the latency was measured as 25.4 microseconds. The latency for the third class consisting of large messages, was 43.3 microseconds. In all the three cases, the asymptotic throughput was observed to be around 5 MB/sec[13]. These measurements were made on the network built using VME-32 backplanes whose maximum bandwidth is 40 MB/sec. If the new VME-64 (80 MB/sec) or the proposed VME-320 (400 MB/sec) are used, the asymptotic bandwidth is expected to be higher proportionally.

5 Programming Environment

A parallel program for PACE+ consists of two components, one that executes on the FEP and the other that executes in each node[11]. These components are normal sequential programs interspersed with calls to communication routines. At present, programming in FORTRAN and C are supported but support for any other language is easy to provide. Access to hard disk is limited to the FEP component that runs like any normal UNIX process.

The FEP and node components of a parallel program are to be specified in terms of file names in a ‘jobfile’. A suitable mapping of the physical nodes to the logical processes in a parallel application can also be specified as part of the jobfile. Typically, a user preprocesses and compiles his jobfile, and then submits the resulting executable to a job manager. Maintaining a queue of submitted jobs, the job manager allocates the required number of nodes for the jobs waiting in the queue and monitors their status. An application may use the entire back-end or only a segment of it. The size of segment has to be a multiple of clusters. The job manager has the functionality

to partition the back-end in terms of sets of clusters depending on the requirements of the applications.

A source level debugging support provides information regarding exceptions that may happen at run time. This information, pertaining to line numbers of erroneous statements in source programs, is printed on FEP's monitors along with the identifiers of the nodes executing the erring node components. Besides spotting exceptions, the debugging support also identifies mismatching communications at the source level and reports the line numbers of the corresponding program statements. All these tools and utilities are collectively called ANUPAM for ANURAG's Parallel Applications Manager which has been validated over the last three years by a number of users of PACE+ systems.

6 Speed-Ups of Some Applications

Several installations of these PACE+ systems are routinely used for solving problems in aircraft design and fluid dynamics. Some of these installations have 32 nodes while others have 24 and 8 nodes. Many parallel Euler solvers have been developed and tested using these PACE+ systems[14,15]. Some of these solvers are now run regularly in order to gather data concerning the suitability of various aircraft frame models in an aeronautics laboratory at Bangalore, India. In another installation at an educational institute, the system is used by research students as well as the faculty for developing codes based on new computational ideas in fluid dynamics.

In order to give an idea of the efficiency of the communication networks of PACE+ systems, speed-ups achieved on some of the parallel applications being run on PACE+ systems are provided in figure (4). The speed-ups are shown here for four different applications, all of them in computational fluid dynamics. These applications solve Euler or Navier-Stokes equations in a discretised volume in three-dimensions. The application denoted in the figure as A has a structured grid of $161 \times 41 \times 41$ points. A maximum speed-up of 29.32 has been

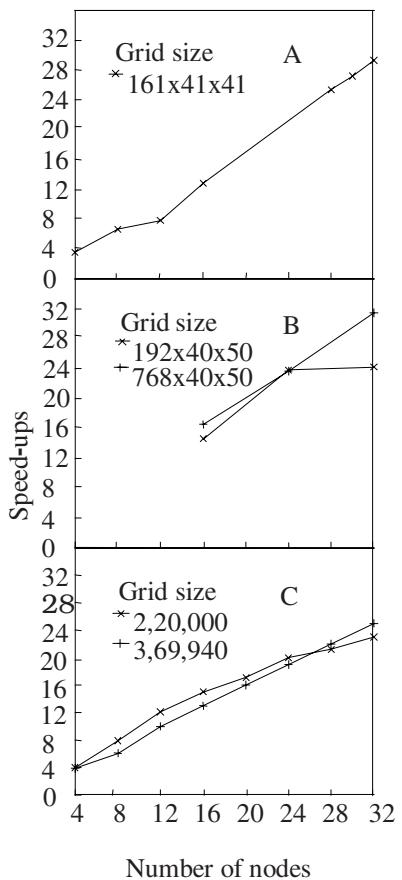


Fig. 4. The speed-ups achieved on PACE+ for some applications in computational fluid dynamics

observed on a 32-node PACE+ system. The application B has been run for two grid sizes. The first one has 192 x 40 x 50 points. Being the smaller of the two, it was observed to have lower speed-ups when compared to the second one of 768 x 40 x 50 points. The latter case gives rise to a maximum speed-up of 31.49 on 32 nodes.

The bottom-most part of figure (4) shows the speed-ups of two different applications labelled collectively as C[15]. One of them corresponds to an Euler solver and the other one, a Navier-Stokes solver. The grid-size of the data domain for the Euler solver is 220,000 grid points and that for the Navier-Stokes solver is 369,940. In both cases, the speed-ups are measured against the time taken in the PACE+ system versus the time taken in a single IBM RS6000/560. The performance of one node of PACE+ is comparable to that of the IBM system. The speed-ups are respectable though there is scope for improvement.

7 Discussion and Future Plans

The implementation of the communication network in PACE+ systems using the concept of memory channel has proved to be an effective one. Firstly the design and development of the communication network turned out to be simple and modular. The network could be easily built-up using openly available, general purpose subsystems such as the VME bus and reflective memories. These subsystems which are commonly used in industrial control applications, turned out to be tailor-made for building a parallel computer based on the memory channel architecture. The VME standard being an industry oriented one, these VME compatible subsystems turned out to be very reliable and rugged making the PACE+ systems very stable for the running of applications non-stop over months.

The main advantage of the memory channel architecture is the low latency in communications. This advantage has also been observed in the usage of PACE+ computers. From a detailed study of the data-transfer rates realisable on PACE+ systems for point to point communications, it has been found that the latency is around 20 micro seconds which is reasonably low. The maximum VME bus bandwidth is about 40 MB/sec. of which, about 5 MB/sec could be realised at application levels on many applications. The fact that these figures are good is borne by the high speed-ups that have been observed on most applications run on PACE+ systems.

In the fast changing scenario that one sees in the field of computers, it is an advantage to have a design wherein adapting to newer and faster processors is very easy and fast. The PACE+ design based on readily available subsystems has this virtue and allows the speedy development of newer versions of PACE+ systems utilising the latest processors that come into the market. Based on the positive experiences in developing and using PACE+ systems, we are now developing a next version of PACE+, called the PACE++. A prototype of this system is currently under development. There will be two communication networks in PACE++, one based on the reflective memories and the other based on high-speed, multi-port switches. The reflective memory network, having a low latency would be used for carrying small messages and flags over long hops. The switch-based network, based on the time-

tested hypercube, would be used for carrying all the large messages. The ANUPAM environment would be upgraded with support for running MPI (Message Passing Interface) standard parallel programs.

References

1. Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salman, J.K., and Walker, D.W.: Solving Problems on Concurrent Processors, Prentice Hall, New York (1988)
2. Kai Hwang and Briggs, F.A.: Computer Architecture and Parallel Processing, McGraw Hill, New York (1994)
3. Zhiwei Xu and Kai Hwang: Early prediction of MPP performance: The SP2, T3D and Paragon experiences, Parallel Computing, Vol. 22, (1996) 917-942
4. Anderson, T.E., Culler, D.E., Patterson, D.A. and the NOW team.: A case for NOW (Networks of Workstations), IEEE Micro, February (1995) 54-64
5. Tandiary, F., Suraj C. Kothari, Asish Dixit and Walter Anderson, E.: Batrun: Utilising Idle Workstations for Large-Scale Computing, IEEE Parallel and Distributed Technology : Systems and Applications, Vol. 2, No. 4, (1996) 41-48
6. Jose Miguel, Agustin Arruabarrena, Ramon Beivide and Jose Angel Gregorio: Assessing the Performance of the New IBM SP2 Communication Subsystem, IEEE Parallel and Distributed Technology : Systems and Applications, Vol. 4, No. 4, (1996) 12-22
7. Richard B. Gillett: Memory Channel Network for PCI, IEEE Micro, Vol. 16, No. 1, (1996) 12-18
8. David K. Kahaner: Parallel computing in India, IEEE Parallel and Distributed Technology : Systems and Applications, Vol. 4, No. 3, (1996) 7-11
9. SPARC RISC User's Guide, Ross Technology Inc., USA (1993)
10. VMIVME-5550 Reflective Memory Board, Document No. 500-035550-000C, VME Microsystems International Corporation, USA (1994)
11. Manuj Sharma et al.: PACE+ User Manual, Report No. ANURAG/PACE+/01, ANURAG, Hyderabad, India (1997)
12. Zhiwei Xu and Kai Hwang: Modelling Communication Overhead : MPI and MPL Performance on the IBM SP2, IEEE Parallel and Distributed Technology : Systems and Applications, Vol. 4, No. 1, (1996) 9-23
13. Manuj Sharma and Anupam Mandal: Communication Overhead Model for Point-to-Point Communications on PACE+, Report No. ANURAG/PACE+/02, ANURAG, Hyderabad, India (1998)
14. Sinha, P.K.: The Solution of Flow Problems Around Complex Configuration Using Euler Code VEEMA on Parallel Computer PACE+, Proceedings of the Third Asian Computational Fluid Dynamics Conference, Bangalore, India, December (1998)
15. Singh, K.P., Biju Uthup and Laxmi Ravishanker: Parallelisation of Euler and N-S Code on 32 Node Parallel Supercomputer PACE+, ADA/DRDO-DERA Workshop on CFD, Bangalore, India, September (1997)

Evaluation of Data and Request Distribution Policies in Clustered Servers

Adnan Khaleel and A. L. Narasimha Reddy

Texas A & M University, adnan,reddy@ee.tamu.edu

Abstract. The popularity of the WWW has prompted the design of scalable clustered servers. Data and request distribution policies greatly impact performance in such servers. In this paper, request response time is used as a measure in evaluating these policies. Our results indicate that policies favoring locality seem to perform better than policies favoring load balance. We propose and evaluate a dynamic client-based request distribution policy.

1 Introduction

In order to support the expected growth, future web servers must manage a multi-gigabyte or a multi-terabyte database of multimedia information while simultaneously serving multiple client requests for data [1]. Similar trends are being observed in traditional file servers because of increasing number of users, increased file sizes etc. Multi-processor based servers and cluster-based servers have been leading candidates for building such servers [1, 3]. In this paper, we will study clustered servers. In a clustered server, a front-end node(s) represents the server and may distribute incoming traffic among multiple back-end nodes. The back-end nodes store and serve the data required by the clients. In this paper we will study the problems in organizing and serving the data among the back-end nodes.

In a “disk-mirrored” system, the back-end servers contain identical data and can service a request without having to access files on another machine. File data may be cached in several nodes at once depending on the manner in which requests are distributed. This may lead to inefficient use of cache space.

In a “disk-striped” system, the available disks divide the entire data set that the server to host. Files maybe partitioned such that portions of it may reside on every disk. Apart from providing better access times (due to the multiple disk accesses that can be performed for a single file) disk striping can improve load balance across disks.

The front-end server represents the gateway to the external clients. One of the tasks of the front-end is to determine which back-end server to forward an incoming request i.e. acts as a request distributor. The request distribution schemes employed can greatly impact system performance. Request distribution policies need to consider the impact of load balance and request locality.

The request can immediately be served by a back-end node if the request is cached in its memory. Since disk accesses take much longer than memory accesses, servers try to maximize cache-hit ratios. Cache-hit ratios can be improved by taking advantage of request locality. This requires that requests accessing same data be served by the same set of servers. This is in contrast to load balancing which also results in better response times.

The paper makes the following contributions: (a) provides an evaluation of the various policies based on request response times rather than server throughput, (b) proposes and evaluates a dynamic client-based request distribution policy and (c) considers the impact of data distribution on the performance of the request distribution policies.

2 Request Distribution Schemes

Round Robin: The requests that arrive at the front-end will be distributed to the back-end servers $1, 2, 3, \dots, n, 1, 2, \dots$ and so on in a n -node machine. This results in an ideal distribution as the requests are equally divided amongst all the back end servers. A disadvantage of this blind distribution is that any cache hits that are produced are purely coincidental. Therefore, in effect, not only is every server expected to cache the entire contents of the web site, we also have unnecessary duplication of data in caches across several back-end servers. The back-end nodes may see different waiting times due to uneven hit ratios. In simple Round Robin scheme, the back-end server's current load is not taken into consideration. A slightly modified version of Round Robin called Weighted Round Robin (WRR) assigns a weight to each server based on the server's current load. Requests are then distributed based on the priority of the weights.

File-based Request Distribution: In a file-based scheme, the file space is partitioned and each partition is assigned to a particular back-end server. This scheme exploits locality to the maximum. As requests for the same file are always directed to the same server and hence only that machine will have a cached copy. It is difficult to partition the file space in a manner such that the requests balance out but then this is very dependent on client access patterns and therefore one partitioning scheme is very unlikely to be ideal for all situations. Load balance is not directly addressed in file based distribution. It is assumed that the partitioning of files is sufficiently good enough to make sure server load is balanced. However, this may not be the case and some files will always be requested more often than others and hence load can be skewed.

Client-based Request Distribution: In this scheme, the total client space is partitioned and each back-end server is assigned to service requests from a partition. This scheme is similar to the popular Domain Naming Service (DNS) scheme employed in Internet servers. Client-based schemes completely ignore the server loads and as a result, unequal load distribution can commonly occur.

DNS name resolution is expected to be valid for a time period controlled by the time-to-live (TTL) parameter. However in practice, TTL values are normally ignored by clients making DNS-based distribution a static scheme [1]. Our proposed approach is based on IPRP protocol [2]. IPRP allows a client to be redirected to another server if the server deems it necessary (to improve load balance for example). It has been shown that clients cannot void this mechanism unlike the DNS-based scheme. This makes this approach truly dynamic. Based on workload characteristics, the server may use small TTLs when load balance is important and may use large TTLs when locality is important. The scheme studied here adapts the TTL value over time based on the perceived importance of locality versus load balance.

Locality Aware Request Distribution (LARD): The issue of distribution being completely ignored in file-based distribution is addressed in a technique referred to locality aware request distribution, LARD [3]. LARD assigns a dynamic set of servers to each file. When a request is received, the front-end checks to see if any back-end server have already been designated for this file. If not, the least loaded server in the cluster is assigned to service requests for this file. Details of the LARD algorithm can be found in [3] Several differences between the simple file-based file-space and LARD have to be noted. In LARD, the partitioning is done on the fly, and even though locality is of concern, it is not restricted to just one back-end server.

3 Simulation Model

In order to test and understand the various distribution schemes, a simulation model of a clustered server system was designed and implemented based on CSIM. Each server in the cluster system was modeled as a separate process and each server has its own hard disk, which was also modeled as a separate process. The front-end and back-end servers are functionally different and hence needed to be modeled separately. Each of the above mentioned servers and disk processes have their own queuing system that allows for requests to wait if the server or disk is busy processing the current request. Since this was a trace driven simulation, the design of the front-end was fairly straightforward. The input trace is read for requests and the designated server is determined based on the distribution scheme.

The back-end servers perform slightly more complex functions and since it has the task of caching the data and have to communicate with their own disks (disk mirror) or the disks of other servers (disk striping) to obtain data on cache misses. Processing costs of cache lookups, network transfers etc. involved in servicing requests are based on actual experimentally measured numbers on an IBM OS/2 based machine. Files are cached at the request-servicing node in both disk striping and disk mirroring systems. In addition, files may be cached at the disk-end of the server in a disk-striping system.

In addition to being able to change the distribution scheme, the simulation model permits the changing of the following system parameters: Number of servers, size of memory, the CPU processing capacity in MIPS, disk access times, network communication time per packet and data organization - disk striping and disk mirroring. Processor capacity is set to 50 MIPS.

Two traces were used in this study: (1) The first trace was produced from a web server and contains about two weeks worth of HTTP requests to ClarkNet WWW server [4]. The trace had over 1.4 million requests from over 90,000 clients and included requests for approximately 24,300 files. (2) The second trace was obtained from the Auspex [5] file server the University of California, Berkeley. This trace was collected over a period of one week and consists of over 6.3 million records, has 231 clients and includes requests for over 68,000 files. The traces we have chosen are representative of the 2 areas where clustered server systems are currently being employed. In both cases, the simulation was run for a sufficient period of time to allow the server caches to "warm-up" i.e. fill completely, prior to any measurements on performance being made.

4 Results

4.1 Effects of Increasing Memory Size

Fig. 1 and Fig. 2 show the impact of memory on NSF workload with disk striping. An increase in memory size increases the available caching area and as a result the cache-hit ratios improve. This results in a less number of disk accesses and a decrease in the response time. This effect is observed in all the distribution schemes to various extents. Since file-based is better at extracting locality, it has higher hit rates. LARD's performance, in spite of having very comparable hit rates to RR, is worse off than RR. This is directly attributable to the server queuing times. Due to the sequential nature in which RR distributes its request, queues rarely buildup. This argument does not hold for locality based schemes where increased temporal locality will result in increased queuing times. Increasing memory size does not alleviate this phenomenon. RR was observed to have perfect load balance while file-based, client-based and LARD had disproportionate number of requests queued at different servers. Locality has a big impact on performance in this workload and hence FB scheme outperforms the others.

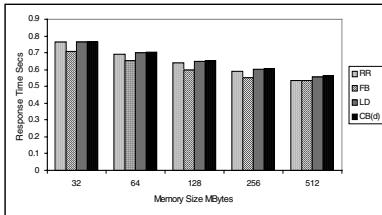


Fig. 1. Response times.

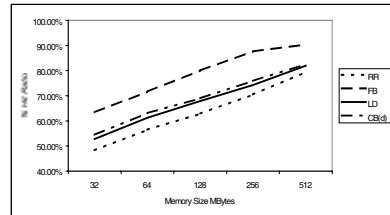


Fig. 2. Cache-hit ratios.

Fig. 3 and Fig. 4 show the impact of memory on web workloads. As compared to the NFS trace, the web trace (studied in these simulations) is characterized by having a comparatively smaller working data set size and a larger number of clients. The consequence of having this smaller working data size is that even low amounts of memory can produce extremely good hit rates. From Fig. 3 and Fig. 4, even at 32 Mbytes, the lowest hit rate is above 90%. A consequence of this is that the distribution scheme plays a less of a role in determining performance. The load distribution across all the schemes is fairly acceptable, the best being RR as expected, and the worst being file-based. With sufficient memory (greater than 128Mbytes), the differences in performance are not significant and all the schemes performed equally well.

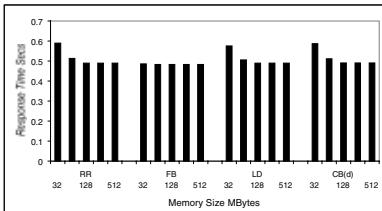


Fig. 3. Web response times.

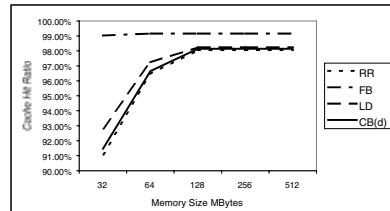


Fig. 4. Web cache hit ratios.

4.2 Scalability

Increasing the number of servers produces interesting results in our simulations. We increased the number of servers from 4 to 8 and 16. The results are given in Fig. 5 to Fig. 6 for NFS trace with disk striping.

In terms of cache hit ratios, RR experiences the least improvement, at 32Mbytes, increasing from being 48.3% with 4 servers to 49.85% for 8 servers and 50.14% for 16 servers. It was discussed earlier that the cache hits that occur in RR are purely probabilistic and this is the reason why the cache-hit rate does not improve. Even though the number of servers has increased, the effective caching area is still equal to that of just one server. At higher amounts of memory (64 - 512MB), the cache-hit ratio actually drops with increasing the numbers of servers. Increasing the number of servers causes a decrease in the amount of locality that can potentially be extracted (by RR) and the probability of file reuse. Locality based schemes do not experience this problem. Server throughput will be improved in all the request distribution schemes with the increased number of servers.

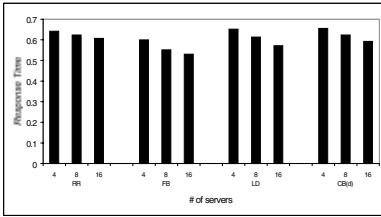


Fig. 5. Response times. vs. # of servers.

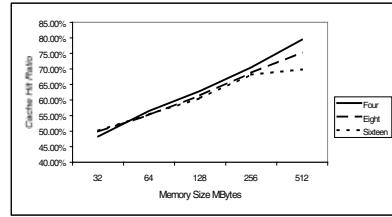


Fig. 6. Cache hit rates vs. # of servers.

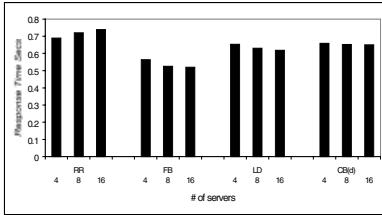


Fig. 7. Response times, disk mirror with 128MB.

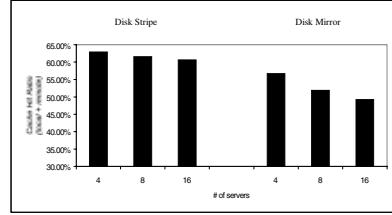


Fig. 8. Cache hit rates for RR, mirroring vs. striping

Fig. 7 and Fig. 8 show the impact of increasing the number of servers with NFS trace and disk mirroring. In contrast to disk striping, RR in disk mirroring experiences a worsening of performance with an increasing number of servers. As the number of servers is increased, the cache hit rates decrease and this leads to higher response times. With disk striping, we observed decreasing hit rates but the response times did not get worse. This is due to the fact that locality can be exploited in disk striping through disk-end caching as observed by higher hit rates in Fig. 8 when compared to disk mirroring. Similar effects were observed in the web trace.

4.3 Disk Striping vs Disk Mirroring

Fig. 9 and Fig. 10 show the cache hit ratios and server and disk queue times under RR in a 4-node NFS server. It should be noted that disk striping due to its automatic distribution of requests produces very low disk queues in both cases. On the other hand, mirroring has consistently higher disk queues. Remote processing of requests and improved hit rates due to remote caching are the reasons for the observed performance characteristics. Remote request processing increases server load in disk striping compared to disk mirroring. Remote hit rates (at the disk-end) reduce the number of disk accesses made in disk striping. Similar results were observed in web workloads.

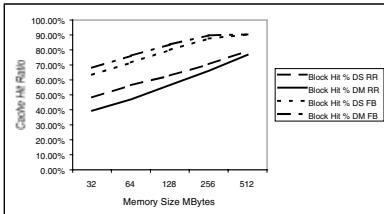


Fig. 9. Comparison of cache hit rates.

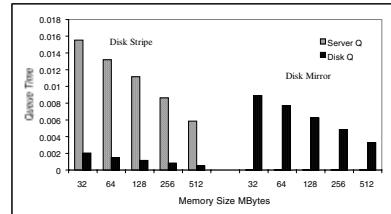


Fig. 10. Comparison of queue times.

5 Conclusion

We evaluated various request distribution schemes and data organization schemes over a clustered server. We used request response time, unlike earlier studies that used throughput, in evaluating the various schemes. For the workloads considered, locality proved more important than load balancing. It was observed that round robin policy, due to its probabilistic nature of exploiting locality, could have worse response times in larger servers. We also evaluated the impact of data organization on servers' performance. Disk striping distributed the requests over several back-end servers and resulted in smaller disk queues. Remote caching in a disk striping system is shown to improve response time due to its exploitation of locality at the disk-end. Dynamic client-based request distribution policy is shown to have performance comparable to round robin distribution. In future, we plan to combine throughput and response time measures in evaluating the server performance.

References

1. Thomas T Kwan, Robert E McGrath, and Daniel A Reed, "NSCA's world wide web server: Design and performance," *IEEE Computer*, pp. 68–74, Nov 1995.
2. S Gupta and A L Narasimha Reddy, "A client oriented ip level redirection mechanism," in *Proc. of IEEE INFOCOM'99*, Mar. 1999.
3. V. S Pai et al, "Locality-aware request distribution in cluster based network servers," in *Proc. of the 8th ASPLOS*, 1998, pp. 205–216.
4. "Web traces and logs," <http://www.cs.rutgers.edu/davison/web-caching/traces-logs.html>; Accessed on 4/11/98.
5. "Auspex file system traces," <http://now.cs.berkeley.edu/Xfs/AuspexTraces/auspex.html>; Accessed on 3/11/98.

Thunderbolt: A Consensus-Based Infrastructure for Loosely-Coupled Cluster Computing

H. Praveen, Sunil Arvindam and Sameer Pokarna

Novell Software Development (India) Pvt. Ltd.,
49/1 & 49/3, Garvebhavipalya, Hosur Road, Bangalore-560008, India
{hpraveen, asunil, psameer}@novell.com

Abstract. This paper presents the design of Thunderbolt, a scalable, reliable distributed computing infrastructure written in 100% Pure Java. The primary goal of Thunderbolt is to make distributed application development easy and intuitive for the programmer. It achieves this by providing structuring mechanisms, such as transactional messaging and group membership that hide the asynchronous, failure-prone nature of the underlying network behind an intuitive and easy-to-use set of Java classes. It implements this using a client-server distributed consensus architecture. Preliminary experimental results are presented. The key contribution of the paper is that it shows consensus to be a practical and effective means of building realistic distributed systems.

1 Introduction

Writing distributed applications is hard! Programmers have to deal with asynchrony, unbounded network latency, node and link failures and network partitions. In the face of this complexity, programmers create centralized solutions that are rarely reusable, and have poor fault tolerance. The principal aim of Thunderbolt is to make structuring abstractions like transactional messaging and group communications available to distributed application programmers. This enables them to concentrate on their application logic while leaving the management of distributed state to the underlying infrastructure. Thunderbolt presents a software-based approach to clustering where Java objects are the units of clustering and the object group abstraction is the principal structuring element. Thunderbolt can be viewed as a “dynamic” clustering infrastructure where Java objects can join and leave a cluster as the application demands. Currently the infrastructure scales well on Intranets. The ideal architecture would be a workstation cluster with or without a high-speed interconnect.

Thunderbolt consists of protocols that solve the distributed consensus[1] problem. Consensus is an abstract distributed agreement problem where objects propose a set of values that need to be decided upon. The protocol ensures that, under certain realistic conditions, the objects eventually agree upon the same value. What is crucial is that most of the distributed protocols that any real application would need to implement such as atomic broadcast, non-blocking atomic commitment, group membership or view synchrony are all simple instances of consensus, thus underscoring its fundamental nature[2].

An efficient solution to consensus implies an efficient solution to these problems as well. Most current work in consensus has been of a theoretical nature. Thunderbolt may be the first system to bring the notion of consensus out of the theoretical literature and into the realm of practical computing.

Thunderbolt has a fault-tolerant[1], client-server architecture[2] where the servers implement the consensus protocol and the clients present a particular consensus problem (e.g. atomic broadcast or view-synchrony) to the servers to solve. This unique partitioning enables the instantiation of various architectural styles ranging from a centralized, single server mechanism as found in Amoeba[3] or ISIS[4], to a fully decentralized peer-to-peer architecture with no single points of failure. This extremely flexible infrastructure enables various reliability versus scalability tradeoffs to be made as demanded by the application.

2 The Thunderbolt Architecture

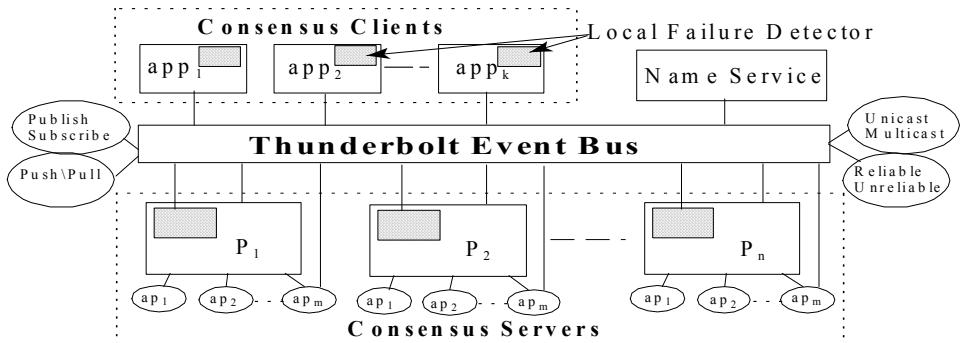


Fig. 1. The Thunderbolt Architecture.

Thunderbolt is implemented using a modular architecture as shown in Figure 1. The Event Bus provides the basic message transport service. The Consensus Service consists of a pool of consensus servers (labeled P_1-P_n). They solve the specific consensus problem (ap_1-ap_m) for the application objects (app_1-app_k). The clients and servers have access to local failure detectors. The Name Service provides unique logical IDs to each object. We describe each component briefly below.

2.1 The Event Bus

The event bus is the message communication channel. It is composed of a stack of protocol objects that can define different Quality-of-Service (QoS) parameters. Currently both unreliable (UDP, IP multicast) and reliable (TCP, reliable multicast) messaging are supported. The reliable multicast protocol is based on gossip-style anti-entropy[5] that is scalable and *eventually reliable* (messages are guaranteed to be delivered if both sender and receiver objects are correct).

Applications need to couple transport reliability with typical messaging semantics like *push*, *pull*, *remote method call* or *asynchronous receive* (callbacks) to implement their required messaging functionality. Receiver objects subscribe for messages based on their Java *types*. Future versions will support content-based subscriptions as well.

2.2 Failure Detectors

On each node, the failure detector, which is necessarily unreliable, monitors the objects of the system and maintains a list of those that it currently suspects to have crashed. Thunderbolt implements the *fail-stop* model: once a process fails, it never recovers. Future versions will implement the more general *crash-recovery* model. The failure detector in Thunderbolt belongs to the class $\diamond S[1]$ and is implemented as defined in [1]. On Intranets and tightly-coupled clusters, the failure detector comes close to achieving eventually perfect properties $\diamond P[1]$, since a failure suspicion means, with high probability, that the object, or the link to it, has indeed failed.

2.3 The Name Service

The consensus servers register with the name service every time a new run of the consensus service is started. The servers are then blocked at a barrier synchronization point until all servers have registered. This helps to implement the fail-stop model where the required number of servers need to be started simultaneously. Then, the name service assigns unique logical IDs to each consensus server in the range 1 to n after which they begin execution.

2.4 The Consensus Service

The consensus service consists of the consensus servers, the consensus clients, the failure detectors and the application protocols. Some of the primary design considerations for the consensus service are:

1. A generic consensus service, decoupled from the specific application protocols is needed.
2. The architecture should be multi-threaded so that some threads can make progress while others are blocked waiting for messages. This makes the service scalable.
3. For a clean separation of functionality, the objects (servers) providing the service should be separated from the objects (clients) using the service.
4. All the application protocols should exploit multicast communication.

On each server, there is a separate thread for each application protocol. There is also a pool of threads that will execute the generic consensus protocol. An application protocol is implemented by defining filter functions[2] that customize the abstract consensus protocol to solve the specific problem. There is a component for each protocol on both the client and server. The client part submits its proposal to the server part to decide upon.

The server part gathers the proposals from all the clients (as defined by the filter function) and then submits them as its proposal to a generic consensus thread to decide. Each such consensus is given a unique ID. For each ID, the corresponding consensus threads on each server then decide upon one value, pass it back to the application protocol thread, which then returns the value to the clients using a highly reliable messaging protocol over the event bus. This separation of the generic consensus protocol from the application protocols makes the architecture modular and flexible. Thunderbolt currently provides implementations of *atomic broadcast*, *non-blocking atomic commitment* and *view-synchrony*[2], all of which use multicast for inter-client communication. The consensus service uses a rotating-coordinator paradigm[1] that can sustain $\lceil n/2 \rceil$ server failures and is therefore highly fault-tolerant. The service also uses multicast for its primary communication step.

3 Experimental Results

Table 1 presents some very preliminary results. It shows the average time for consensus to be achieved on 1 to 5 servers running on a 10MBPS LAN. The heartbeat timeout is 1.5 seconds. The code is unoptimized.

No. of Servers (Pentium III, 450 MHZ, Win95, 64 MB RAM, 6.4 GB Hard drive)	Avg time/consensus (ms) (for 1000 consensus rounds)
1	12.4
2	25.2
3	38.4
3 (coordinator killed)	2004
4	50.2
5	64.5

Table 1. Consensus run-time results.

3.1 Observations

1. The results on a single processor can be far superior since the server can return a decision immediately without executing the full protocol.
2. There is a linear increase in time per consensus as the number of servers increase. This is encouraging, suggesting a practical use as a clustering manager where an appropriate number of servers can be deployed for fault tolerance.
3. When the coordinator is killed, the servers take 1.5 seconds (the heartbeat timeout) to sense its failure. Then, the next logical server takes over as coordinator and completes the protocol. In such a situation it takes at least 2 seconds per consensus round. In a tightly coupled cluster the heartbeat timeout can be set more aggressively, thus providing faster failover.
4. Consensus is a highly communication-intensive protocol. By connecting the servers over a 100MBPS LAN or a high-speed interconnect such as Myrinet (640MBPS), HiPPI or VIA, very high performance can be obtained.

4 A Clustering Application

Using the application protocols that are currently implemented, it is particularly simple to build a clustered, highly available service. We have constructed a fault-tolerant version of the Java Message Service (JMS) as follows: every operation and message sent to the primary server is replicated on the secondary servers using *atomic broadcasts*. Both the primary and backup servers form a *view-synchronous* group; the information about the current group membership is circulated to the clients as well. When the primary server fails, the clients automatically fail over to the secondary server after a few seconds. Since the entire state on the secondary is an exact replica of the state on the primary, the clients restart exactly where they left off.

5 Conclusions and Future Work

Consensus is a local-to-global state transformer that provides an elegant and powerful means of coordinating objects in a distributed system. In environments that permit accurate failure suspicions (e.g., a closely-coupled cluster), its behavior closely approximates a virtually-synchronous system like ISIS. But in environments where failure suspicions are error-prone such as in lossy networks, the protocol maintains liveness, unlike ISIS, which tends to thrash in such situations. Since it leverages multicast (ISIS and Amoeba use point-to-point communications), consensus will scale to future generation networks such as Ipv6. Finally, the service itself has a high degree of fault-tolerance. However, there are several optimizations that need to be incorporated before consensus can be deployed in a real setting. A few of them are:

1. Implement the crash-recovery model for consensus.
2. Make both the heartbeat protocol and the client-server interactions hierarchical, thus minimizing unnecessary messages while improving scalability.
3. Implement a fast consensus protocol to handle speed mismatches between servers while minimizing the latency degree.

With these optimizations and/or a higher-speed interconnect, Thunderbolt can provide an alternative and eminently practical substrate for reliable distributed computing.

References

1. T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. JACM, 34(1), pages 225-267, March 1996.
2. R. Guerraoui and A Schiper. The Generic Consensus Service. TR 98/282, EPFL, 1998.
3. F. Kaashoek and A. Tanenbaum. Efficient Reliable Group Communication for Distributed Systems, Amoeba Project Publication, 1994.
4. K. Birman and R. van Renesse. Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society Press, 1993.
5. R. Golding. Weak-consistency group communication and membership. UCSC TR UCSC-CRL-92-52, Dec. 1992.

Harnessing Windows NT for High Performance Computing

A. Saha

K. Rajesh

Sunita Mahajan

P. S. Dhekne

H.K.Kaura

Computer Division, Bhabha Atomic Research Centre,
Trombay, Mumbai 400 085.
Email: rajesh@magnum.barc.ernet.in

Abstract. Parallel processing has emerged as a cost-effective solution for high performance computing. The “cluster of workstations” approach to parallel processing has also gained widespread acceptance. The ever increasing speed of the microprocessor and the availability of economically viable high speed interconnection networks combine to offer a powerful and inexpensive solution to parallel processing using a cluster of workstations. Windows NT offers a number of features that can be utilized to build such a system without compromising on functionality. In this paper, we describe the development of such a system. We also describe the development of the parallel processing library and related tools for parallel program development..

1. Introduction

There has been an ever-increasing demand for computing power, both for general as well as for scientific and engineering applications. Even as processor technology becomes more advanced and faster processors are available, the complexity of problems that need to be solved, and thereby the size and complexity of the programs written for the solution also increases. This implies that the demand for more computing power will always be there. Parallel processing uses the power of multiple processors to perform simultaneous calculations to achieve this goal.

Large number of users could benefit from advances in parallel processing if developmental efforts in this field are also concentrated on operating systems like Windows 95/98/NT which have a huge user base. Windows NT offers an environment that can be used to develop parallel systems, which was hitherto the domain of UNIX. Windows NT is a robust, portable, and secure operating system which justifies itself as an operating system of choice for designing a parallel system. It also has the same user interface as Windows 95/98.

The Winsock interface in Windows allows the development of portable message passing libraries using TCP/IP. The Win32 API supports security and debugging, aiding in the development of tools for supporting parallel program development. Several optimizing compilers are also available on this platform permitting fast code

execution. Development of graphical support tools also becomes easy, as various visual programming environments are available in NT.

A message passing library, ANULIB, has been developed that exposes the calls for communication among processes comprising the parallel program. This has the same interface as ANULIB developed on previous versions of ANUPAM [1][2]. Hence, already existing user code running on previous versions of ANUPAM parallel systems and not making use of non-portable system calls run without any modification [1][2][5]. PSERVER, a service under NT, was developed to facilitate the spawning of processes in the machines configured to run the parallel program. A stack trace utility, PTRACE was developed to facilitate debugging of parallel programs. A syntax analyzer, SYN, allows the developer of parallel programs to verify whether the program uses the syntactically correct ANULIB calls. Apart from this, standard message passing libraries like PVM and MPI can also be used on the system.

Performance results are given that have been taken on a cluster of four machines based on Intel PII @266 MHz with 256 MB RAM, and interconnected by a 100 Mbps Fast-Ethernet switch. One of the systems runs Windows NT Server and the rest run Windows NT workstation.

2. Platform Considerations

Users intent on developing code for the Windows environment and coming from the Unix domain need to take into account some differences between these platforms. These are also valid while migrating code.

There are some differences between the paradigm of process interaction between Unix and Windows. For instance, unlike Unix, signals cannot be used for inter process communication under Windows. Calls like 'fork()' are not implemented in the Win32 API. Parent-child relationships are also not maintained under Windows. Processes that spawn other processes must keep track of child process handles. Graphics using X-Windows requires complete rewriting for the Win32 API. There are also differences in the implementation of file system, memory management and security API.

Windows NT is a better choice for an operating system in a parallel machine compared to Windows 95/98. There are features that are not supported by Windows 95/98 compared to Windows NT. Security API and SCM (Service Control Manager) are not implemented under Windows 95/98 and they lack in robustness compared to NT. NT allows the development of services, akin to daemons on UNIX. Windows 95/98 is available only on the Intel platform whereas NT is available on multiple platforms. In a multi-user environment some facilities like security, disk-quotas etc. become necessary. Though it is possible to have a Windows 95/98 based system with minor modifications in the software, we then have to compromise on other aspects such as security etc.

3. ANULIB

ANULIB is the message-passing library that has been developed using the Winsock interface for TCP/IP. The library was developed mainly to support the in-house parallel applications that were originally developed on the ANUPAM parallel computer in BARC. The library uses TCP to maintain communication channels between the processes and to transfer data between them.

Error handling routines in the library guarantee that the user is intimated immediately of any error encountered in the parallel program and that the resources being used are freed. The library creates a separate thread to listen on an 'error' port. Before a process terminates abnormally it sends a packet to all the other processes on this port. The processes terminate gracefully after informing the user of the error. Actual error in descriptive textual format allows the user to track down the cause of the error.

A signal handler in the library handles the signals that are generated. However, not all signals on Unix are raised on NT. The library guarantees proper transfer of data by internal checking in the send/receive calls.

4. Parallel Server (PSERVER)

The parallel execution server has been developed to facilitate the spawning of slave processes and runs on all the machines in the parallel system. The library contacts the server, which then spawns the slave program. The service runs under a domain wide account and has privileges that allows it to connect a drive letter to the home-directory of the user. The server sets some environment variables and spawns the slave program. It then closes the handles to the spawned process and its primary thread so that the system can claim all the resources when the slave exits.

PSERVER has been developed as a service under NT using calls to the SCM (Service Control Manager). It is therefore possible to control it using the 'services' applet in the control panel. It is started automatically at system boot-up time and is automatically stopped at system shutdown by NT.

5. Portable Executable Trace (PTRACE)

PTRACE allows a user to have a trace of the call stack, from the point an exception was generated, to the startup routine. The line number(s) and the source file name(s) are also generated in the trace.

The executables for Windows are in the PE (Portable Executable) format [6] which is very similar to the UNIX COFF (Common Object File Format).

PTRACE first verifies that the executable contains a valid NT signature and is built for the same machine architecture. It also checks whether the line number and symbol table information are present. It then reads the executable file to fill data structures with line number, file name, and exported function information. PTRACE then informs the system that all errors should be notified, i.e. debugging events are

generated for all exceptions. In the event of an exception, it walks up the stack using the frame pointer information from the thread context and reading from the debugged process' memory. The context contains the state of various registers and is different for processors from different families. The stack walking code also needs to be modified for the different processors.

Win32 base debugging API was used to control the execution of the programs. PTRACE makes it possible to monitor the creation and termination of processes and threads, access violations, floating point denormals and inexact, integer and floating point divide by zero, stack overflow and a number of other exceptions. However, by default, Win32 initializes the floating point unit not to generate exceptions. Hence, it is necessary to initialize the floating point unit accordingly at the start of the program. This is done by linking with an object module that initializes the floating point unit at run time by writing a control word to it.

6. Syntax Analyzer (SYN)

It is an utility for detecting syntactic and semantic errors in Fortran parallel programs written using ANULIB library calls. It detects some common errors in syntax and semantics in the use of ANULIB library calls in a parallel program. ANULIB message passing calls have an argument for the type of the data to be sent. SYN checks that the number of arguments passed is correct, that the types of the arguments are correct, and that the type of data being sent is actually the same type as the one being put in a call to the library.

7. Performance Results

Nproc	MFLOPS	Speedup
1	25.73	-
2	48.07	1.87
3	70.5	2.74
4	91.82	3.56

Table 1. Linpack Performance

Nproc	Time(Secs.)	Speedup
1	126.3	-
2	65.32	1.93
3	45.1	2.8
4	35.28	3.58

Table 2. VASBI timings

Nproc	Time(Secs.)	Speedup
1	1139	-
2	572	1.99
3	382	2.98
4	286.8	3.97

Table 3. MOLDY timings

LINPACK is a standard benchmark for parallel machines. It solves a dense system of linear equations. Timings were taken for a 1000x1000 double-precision matrix.

VASBI (Viscous Analysis of Symmetric Bifurcated Intake) is a computational fluid dynamics code and is an excellent example of geometric parallelism [3]. It is a finite volume code that is used to compute the compressible turbulent viscous flow through symmetric bifurcated ducts.

MOLDY (Molecular Dynamics) deals with molecular dynamics simulation [4].

8. Concluding Remarks

Programs like LINPACK that involve a lot of communication can scale better by having faster interconnection networks. However, even with prevalent interconnection technology and machines it is possible to solve the grand challenge problems.

Windows NT offers an alternative environment for parallel processing using ordinary desktop computers. Even with commodity microprocessors it is possible to achieve supercomputing performance. Moreover, this gives a larger fraction of supercomputing power at a much lower fraction of the cost of traditional supercomputing platforms. Parallel processing can now reach a vaster audience using PCs running widely used operating systems such as Windows NT.

9. References

- [1] Sunita Mahajan, P.S. Dhekne, et al., "ANUPAM Programming Environment", First International Workshop on Parallel Processing, Dec'1994.
- [2] Sunita Mahajan et al., "ANULIB – An Efficient Communication Library for BARC's Parallel Processing Systems", Seminar on Advances in Computing, Apr'1998.
- [3] Biju Uthup et.al., "Computation of Viscous Compressible Turbulent Flows through Air-Intake on the Parallel Computer BPPS-24", Proceedings of ECCOMAS-94, Sept'1994.
- [4] S.L.Chaplot, "Molecular Dynamics Computer Simulation - Parallel Computation", Proceedings of the seminar SSV-94, Feb'1994.
- [5] P.S. Dhekne et. al., "Anupam – Alpha, BARC's Solution for High Performance Computing", Proceedings of the 3rd HPC Asia'98 conference, Singapore, Sept'1998.
- [6] Randy Kath, "The Portable Executable file format from top to bottom", Microsoft Developer Network.

Performance Evaluation of a Load Sharing System on a Cluster of Workstations

Yanal Hajmehoud, Pierre Sens, and Bertil Folliot

Laboratoire d'Informatique de Paris 6

Université Pierre et Marie Curie

4, Place Jussieu, 75252 Cedex 05, FRANCE

[yanal,sens,folliot]@src.lip6.fr

Abstract. This paper presents a methodology to evaluate the performance of load allocating algorithms using queuing network facilities. First, we develop a generic model of load sharing systems. Then, a specific model is derived from the generic model that simulates a real load sharing system. This phase consists of successive manipulations of algorithms parameters until reaching satisfied approximations. Once the suitable parameters have been tuned, which configure the specific model, it is easy to evaluate the performance of the load allocating algorithms for various applications and hardware configurations.

1 Introduction

To maximize the performance in a distributed-computing environment, it is essential to evenly distribute the load among the hosts. Many researchers have demonstrated that simple load sharing algorithms produce considerable performance improvement [4]. Classical algorithms are *CPU load balancing* where processes are transferred across the network from highly loaded host to a lower loaded host. In multi-criteria approaches, some heuristics propose to take into account both application behaviors and the global state of the system [1,2]. The main difficulty of such algorithms is to find the appropriate parameters geared to each application such as load thresholds or the weight of various criteria.

We propose a configurable model to help the design of process allocation algorithm. In a first step, we develop a generic model of load sharing systems. Then, in the configuration step, we specialize this model according to a static system configuration. In the tuning step, we evaluate performances of several applications and we compare these measurements with the real measures obtained from an existing migration system. We refine parameters of the specific model until getting simulated performances that are very close to real measurements. Once the simulator is calibrated, we change heuristics of load balancing strategies and when a worthwhile optimization is found, the latter is integrated in the real system.

The remainder of this paper is organized as follows. The next section outlines common load sharing policies and presents the GatoStar system which we have chosen to tune our simulator. The Section 3 describes the model. Finally, the Section 4 presents our performance evaluation study.

2 The Load Sharing System

GatoStar is a load sharing facility [2], that we have developed previously at the University of Paris 6. It automatically distributes parallel applications among hosts according to multi-criteria algorithms. A migration mechanism is periodically activated to supervise the load balancing in the network.

GatoStar was built on top of Unix with no modifications to the UNIX kernel. It works on a set of heterogeneous workstations connected by a local area network. GatoStar defines migration domains of compatible hosts, where processes can freely move. The basic architecture consists of a ring of hosts which exchange information about hosts' functioning and processes' execution.

To take into account the heterogeneous features of hosts, the load of a host is defined as directly proportional to CPU utilization and inversely proportional to its CPU speed [5]. On each host, the load sharing manager locally computes an average load and uses the ring to exchange load information.

Processes are allocated to the most lightly loaded host, only if the load of this host is below the *overload threshold*. Otherwise, a process is started on its originating host as all hosts in the system are highly loaded, or used by interactive users.

To react to applications and environment evolution, GatoStar includes a process migration mechanism. Environment evolution results from foreign activities : an interactive user logs in or sends a remote job to an already loaded host. Application evolution stems from variations in resource usage (processor occupation, communication between processes, and memory demands).

The migration decision is based on two opposite load thresholds : *migration and reception thresholds*. The load sharing manager periodically verifies two conditions : (1) a faster workstation load is under the reception threshold, (2) the local host load is above the migration threshold and at least one workstation load is under the reception threshold. The former condition allows the potential use of fast available hosts, and the latter one allows to decrease the overload of the local host. If one or both conditions are true, the manager of the loaded host chooses a process that has been executed locally for at least a given slice time. This prevents short-lived process to be migrated and also prevents process thrashing.

3 The Model

The developed model is a network of *service stations*, constructed using the QNAP2 package utility. We have distinguished four principal types of activities inside each host of the system. The first activity concerns the injection of programs by the host's owner. The second one concerns the distribution of processes derived from the user programs among the network. The third one concerns the execution of the processes at the host. The last one concerns the interface between the host and the network. We have gathered these activities into separated modules. Therefore, each module has a well known functionality which is defined

by a set of algorithms with their appropriate parameters. The generic property of the model comes from the implementation of several algorithms inside each model. At the configuration phase, the programmer chooses his/her algorithms and the values of their parameters. For example, the programmer can choose the load sharing algorithms (cyclic, random, specific algorithms) for the load distribution module.

A host of the system is described by five modules: a *program generator* (that generates parallel applications and sequential programs simulating the utilization of the host by his/her owner), a *local clock*, a *distributing manager* (that simulates the load sharing algorithms), an *executing manager* (that simulates the processors, disks, memory and communication operations) and finally, a *communicating manager* (that simulates the interface between the site and the network). The LAN is represented by one queue with one or several servers. The queue length reflects the load in the network. The number of servers reflects the accessibility of the network by the hosts.

We generate sequential programs, and parallel applications. A sequential program consists of one task which is executed in the local host and has no interaction and communication with the other tasks of the host. This type of programs (called *local tasks*) simulates the commands launched by the local user of the workstation (ls, lpr, etc). The execution time of a local task is divided in *CPU-segment* and *I/O-segment*. The CPU-segment represents the execution time on the processor and the I/O-segment represents the I/O operations on the disk. The generation rate, the execution time and the ratio between the CPU-segment, and the I/O-segment are parameters.

A parallel application is composed of several tasks (called *parallel tasks*) which can be executed in parallel. Parallel applications are defined by their precedence graphs. A parallel task is composed, in addition to the *CPU-segment* and the *I/O-segment*, of a *Comm-segment* which simulates the communication with the other tasks of the application. The user determines the sizes of the segments.

To tune the parameters of this generic model, we have performed successive manipulations of the model parameters until reaching results almost similar to those obtained by the real load sharing system. We have compared many executions on both our simulator and the real system including computation, I/O and communication operations. The measures have been carried out for a network of 8 heterogeneous hosts equipped with GatoStar. We have measured that the difference between the two sets of measures is less than 10%.

4 Performance Study

In this section, we study the impact of load sharing strategies and their parameters on the system performance. We concentrate this study on the values of the load thresholds, the load balancing policies, and the weight of I/O operations compared to communications over a network.

4.1 Thresholds Values

Thresholds are key features and directly influence the performance of load balancing strategies. We have compared two policies of the thresholds. In the classical one, which is implemented in GatoStar, allocation and migration decisions are based on fixed thresholds. In the adaptive one, thresholds depend on the global system state (the current global load and the hosts characteristics).

Figure 1 compares the performance of these two policies. We have executed two parallel applications : application A is composed of 40 tasks that have identical execution times, application B is composed of 48 tasks that have different execution times (24 short tasks and 24 long tasks). We measured the speedup compared to a local execution for the two policies. We note that the adaptive strategy always improves the performance of the network, particularly for heterogeneous application (up to 30% for application B).

This study has encouraged us to implement the adaptive strategy in the GatoStar system. Similar remarkable benefits have been observed in the real environment.

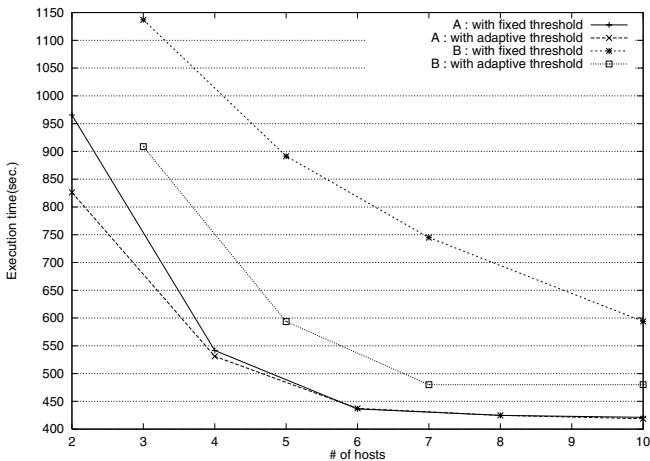


Fig. 1. Adaptive thresholds.

4.2 Placement vs. Migration

To investigate the additional benefits of the migration over the initial placement, we have compared four load sharing strategies : two adaptive load sharing strategies (placement with and without migration), and two static strategies (random and cyclic which is the PVM load sharing algorithm).

Figure 2 shows the speedup of the four strategies for a parallel application composed of 30 identical tasks as a function of the hosts number. We observe

that the adaptive load sharing strategies improves the performance over static approaches (as random and cyclic). Moreover, migration balances the load more evenly than placement.

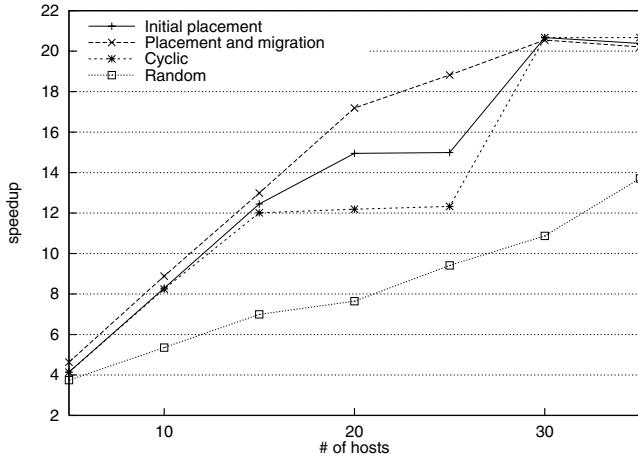


Fig. 2. Comparison of load sharing policies.

For the majority of the tested applications, we found that the additional gain of the migration over the initial placement varies between 15 % and 35 % (depending on the application and the network configuration). Therefore, the combination of both strategies is particularly important.

4.3 Impact of Communication and I/O

The majority of the studies of the load balancing problem consider applications using only the CPU resources. Few studies show the influence of communication or I/O on the overall performance [3]. To investigate this influence, we have executed two parallel applications composed of 20 tasks, on a network of 10 hosts. Each task of the first application contains *CPU-segment* and *I/O-segment*. Each task of the second application contains *CPU-segment* and *Comm-segment*. Figure 3 shows the speed-up as a function of the ratio of communication over computation for the first application and the ratio of I/O over computation for the second application.

We note that the first application has a higher speed-up than the second one. Therefore, the communication has more important impact on the performance than I/O impact. Consequently, the communication criterion must have a higher priority than the I/O in a multi-criteria load sharing heuristic.

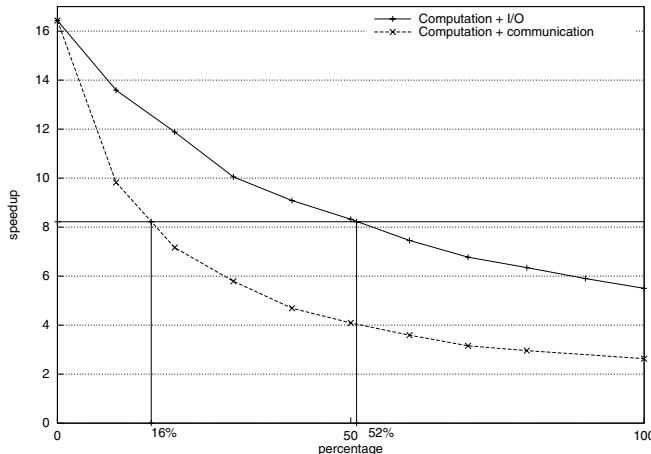


Fig. 3. Communication and I/O.

5 Conclusions

1 We presented a methodology to evaluate the performance of load sharing algorithms. We began by designing a generic model of load sharing systems, then we derived a specific model that mimics a real system.

Among many results obtained, we highlighted the importance of the adaptive thresholds policy against the fixed thresholds policy. We measured the additional benefits of the migration mechanism over the initial placement. However, the maximum additional amelioration observed is about 35%. We also concluded that the communication operations have more important impact on the overall performance than I/O operations.

References

1. A. Barak and O. La'adan. The mosix multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4), 1999.
2. P. Sens and B. Folliot. Gatostar : A fault tolerant load sharing facility for parallel applications. In *Proc. First European Dependable Computing Conf.*, pages 581–598, 1994.
3. J. P. Singh, E. Rothberg, and A. Gupta. Modeling communication in parallel algorithms: A fruitful interaction between theory and systems? In *In Proceedings of the 6th annual ACM symposium on parallel algorithms and architectures*, 1994.
4. Y. T. Wang and J. T. Morris. Load sharing in distributed systems. *IEEE Transactions. on Computers*, c-34:204–217, 1985.
5. S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software-practice and experience*, 23(12):1305–1336, 1993.

Modeling Cone-Beam Tomographic Reconstruction Using LogSMP: An Extended LogP Model for Clusters of SMPs

David A. Reimann¹, Vipin Chaudhary², and Ishwar K. Sethi³

¹ Department of Mathematics, Albion College, Albion, Michigan 49224

² Department of Electrical and Computer Engineering, Wayne State University, Detroit, Michigan 48202

³ Department of Computer Science, Wayne State University, Detroit, Michigan 48202

Abstract. The tomographic reconstruction for cone-beam geometries is a computationally intensive task requiring large memory and computational power to investigate interesting objects. The analysis of its parallel implementation on widely available clusters of SMPs requires an extension of the original LogP model to account for the various communication channels, called LogSMP. The LogSMP model is used in analyzing this algorithm, which predicts speedup on a cluster of 4 SMPs using 10 Mbps, 100 Mbps, and ATM networks. We detail the measurement of the LogSMP parameters and assess the applicability of LogSMP modeling to the cone-beam tomography problem. This methodology can be applied to similar problems involving clusters of SMPs.

1 Introduction

Tomographic reconstruction from projections using computed tomography (CT) is the non-invasive measure of structure from external measurements. The information obtained describes both internal and external shapes and material densities. This is particularly useful when one cannot make internal measurements on the object of study for a variety of reasons. These reasons might be cost, no known non-invasive technique, or no physical means to make internal measurements.

With cone-beam CT, a set of two-dimensional (2D) planar projections, consisting of $N_u \times N_v$ pixels, are acquired at equal angles around the object. These projections are filtered and backprojected into a volume of $N_x \times N_y \times N_z$ voxels. Let N_θ be the number of projections acquired. Cone-beam tomography systems are useful in assessing microstructure of biomedical and industrial objects. Tomography applications continue to grow into areas such as reverse engineering, quality control, rapid prototyping, paleontology, geology, and nondestructive testing. Cone-beam tomography systems offer greater scanner efficiency and image quality, but require much more computing [1]. To improve reconstruction time, a parallel algorithm was developed using the MPI library for communications [2,3]. The parallel algorithm is based on the serial algorithm by Feldkamp [4]. We have optimized this algorithm for a cluster of SMPs.

This processing requires roughly $O(N^4)$ time to reconstruct an image volume of N^3 voxels. For large image volumes, the reconstruction time on a serial computer far exceeds the acquisition time. This is of particular importance as the desire to resolve more detail in larger fields-of-view has demanded increased image sizes. Objects are routinely reconstructed into image volumes of 512^3 voxels and a strong desire to use 768^3 , 1024^3 , and larger volumes in the future.

The original LogP model [5] was extended to clusters of SMPs, called LogSMP. The LogSMP model is used in analyzing the parallel cone-beam reconstruction algorithm to predict speedup on a cluster of 4 SMPs using 10 Mbps, 100 Mbps, and 155 Mbps ATM networks. We detail the measurement of the LogSMP parameters and assess the applicability of LogSMP modeling to the cone-beam tomography problem.

2 LogSMP Model

The LogP model characterizes a homogeneous parallel architecture with bulk parameters and assumes no specific topology [5]. This model contains 4 parameters: the communications latency L , the processor overhead required for sending or receiving messages o , the bandwidth/minimum gap between successive messages g , and the number of processors P .

When using a cluster of SMPs, one must account for the differences in intra- and inter- SMP communications. The LogSMP model accounts for this additional communication channel. For this discussion, assume there are q SMPs and let SMP_1 contain the root node. Let P_i be the number of processors on the i th SMP, such that $\sum_{i=1}^q P_i = P$. Since P_i processors are on the i th node, their intra-communication is governed by the LogP parameters for that channel, namely g_i , L_i , and o_i . The SMPs can communicate with each other as dictated by the LogP parameters for the inter-communications channel, namely g_0 , L_0 , and o_0 . In the degenerate case where there is only one processor on the i th SMP, the parameters g_i , L_i , and o_i are of no consequence. The g values are based on the fastest processor in a heterogeneous environment. This is shown in Fig. 1.

3 Parameter Measurements

Modeling performance involves assessing several timings when using the LogSMP models. The network speed relative to processor speed is a vital part of the modeling. The parameter g , in instructions/byte, measures the number of instructions which can be processed in the time to send one byte of information between two processors. Processor speed and communications speed can be used to derive g . In addition, network latency and overhead must be measured.

For the experiments, a cluster of four Sun Enterprise SMPs, each running SunOS 5.6, was used. The first had six 250 MHz nodes and 1.5 GB total RAM. This machine was used for processors 1–6 in the experiments. The other three SMPs each had four 250 MHz processors with 0.5 GB total RAM on each SMP.

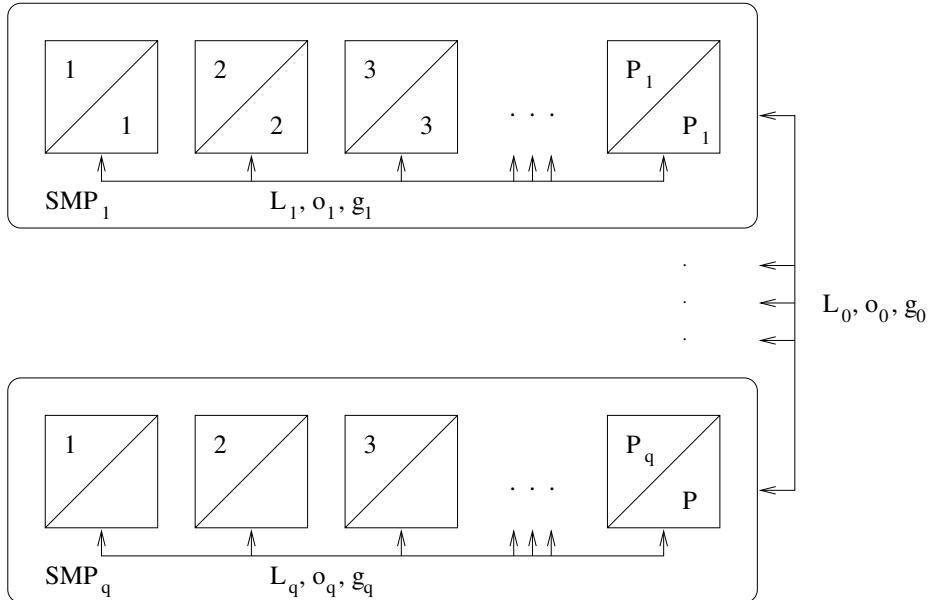


Fig. 1. In a cluster of q SMPs, the LogSMP parameters consists of the LogP parameters are required for intra- and inter- SMP communication. Each SMP can be considered a subgroup with a particular local root processor in addition to the overall root processor

Either a 10 Mbps Ethernet, 100 Mbps Ethernet, or a 155 Mbps ATM link was used for the communication. These architectures will be designated wsu10, wsu100, and wsuATM in this discussion. Version 1.1 of the MPICH implementation of MPI was used.

The processors performance was analyzed by measuring execution time of problems of size N and one processor and timing the backprojection steps. The times were then compared to the single processor model floating point operations. The processor speed was determined to be 45.1 million floating point operations per second using an independent benchmarking program. Results from this are consistent with other benchmarking programs.

Communications speed and overhead was measured using round trip point-to-point passing of messages having various power of 2 sizes from 0 to 4 MB in length. The MPI routines `MPI_Send` and `MPI_Recv` were used. For each message size, 1000 trials were measured to reduce the variance in the estimate.

The parameter g was determined by dividing the processor performance, measured in instructions per second, by the communications speed, measured in bytes per second. While the value g will in general be a function of the message size, the model presented uses the peak value. The resulting values are 0.587, 3.31, 4.59 and 39.2 cycles for the SMP, wsuATM, wsu100, and wsu10 respectively.

The latencies were 91.2, 89.1, 90.3, and 91.5 cycles for the SMP, ATM, 100 Mbps Ethernet, and 10 Mbps Ethernet respectively. In each case the processor overhead was assumed to be 0.1% of the communication time.

The overhead for starting a parallel MPI program was also determined. For each network, the influence on the number of nodes used in the computation was determined by running a parallel program which terminated immediately upon startup. A program consisting of `MPI_Init` followed immediately by `MPI_Finalize` was used. The time to execute this program was repeated ten times for 1 through 18 processors. The times from using 1 through 10 processors was fit to a line and used to predict startup times in the model. The time increases by about 0.5 seconds per processor for the wsu10, wsu100, and wsuATM networks. The variation and magnitude in startup time varies significantly once more than ten processors (more than 2 SMPs) are used.

Another significant factor in the total time required is due to reading raw projectional data and writing the final reconstructed image. The amount of data read is $2N_\theta N_u N_v$ bytes, or for the simplified problem of size N , the total amount is πN^3 bytes. For writing the final data, $2N^3$ bytes are written. The times for input and output were empirically measured and incorporated into the model.

4 LogSMP Model for the Parallel Cone-Beam Algorithm

A voxel driven approach is taken where the volume is distributed over the processors and each projection is sent to every processor. Each processor sees every projection, but only a small subset of the reconstructed voxels. The total memory required for this implementation is approximately equal to the total number of voxels, which is just the product of the volume dimensions and the bytes required for each voxel (4), namely $4N_x N_y N_z$. The voxel data is not replicated on each processor, so individual memory requirements are not as demanding. Another advantage of this method is that the data is acquired in a serial fashion and processing could be done in concert with acquisition.

The parallel algorithm utilizes a master processor which does all I/O and essentially dictates the tasks performed by the other processors. While parallel I/O is sometimes available, the algorithm requires the final reconstructed data to be explicitly sent back to the master processor. The MPI library is initialized using `MPI_Init`. MPI assigns each processor in the communication group a unique id. The number of other processes in the group is also available. All processes are initially only part of the `MPI_COMM_WORLD` process group. An MPI communicator is then created for each SMP containing all processes on that SMP. Another communicator is created containing the root nodes of each SMP communicator group.

For the case of $N = N_x = N_y = N_z = N_u = N_v$, the number of projections N_θ should be $\frac{\pi}{2}N$, and thus the complexity of the problem is $O(N^4)$. N can also serve as an upper bound when N is the maximum of those parameters. Using this simple model it becomes easier to study the effects of N , P , and g on the theoretical speedup.

In addition to computation requirements, a major nontrivial aspect of the problem is the data size required. For example, an acquisition of 804 views of 512^2 images with 16 bit integer pixels is needed to create a 512^3 volumetric data set with 32 bit floating point voxels. The use of 32 bit floating point voxels is required to provide sufficient accuracy in the result. The memory requirement to store the entire volume is 512 MB, which is currently feasible on SMPs, but even more so on a collection of SMPs.

The time required to perform a reconstruction can be expressed as

$$T = T_0 + T_i + T_r + T_f + T_c + T_b + T_g + T_w \quad (1)$$

where T_0 is the MPI startup time, T_i is the initialization time, T_r is the time to read a projection from disk, T_f is the time to filter a projection, T_c is the time to communicate the filtered projection from the root processor to the others for backprojection, T_b is the time to backproject a filtered projection, T_g is the time required to gather the reconstructed volume to the root processor for subsequent scaling and writing, and T_w is the time to write the reconstructed volume to disk. In general, these times are functions of processor speed, network bandwidth and latency, and the number of processors used. T_0 , T_r , and T_w are measured empirically as a function of P as described below.

The algorithm consists of repeated asynchronous sends of the filtered projections from the root node a single node in each SMP, and synchronous broadcasts occur from that node to all others in that SMP. The root node also accounts for the filtering time and reduces the number of voxels to backproject by the corresponding amount. In this algorithm times can be modeled as follows. $T_i = 7N_u N_v$ is the initialization time and consists of memory allocation, and precomputation of values required for weighting the projectional data. $T_f = N_\theta(N_u N_v(3 + 18 \log_2(2N_u)))$ is the time required to filter a projection on the root processor. $T_b = N_\theta(N_y N_x(13 + 12N_z))/P$ is the time required to back-project on P processors. $T_g = (P_1 - 1)4N_y(N_x N_z g_1 + L_1)(1 + o_1)/P + (P - P_1)4N_y(N_x N_z g_0 + L_0)(1 + o_0)/P$ where the first expression is gathering on root SMP, and second expression is gathering from remote (non-root) SMPs.

$$T_c = N_\theta(4(P_0 - 1)(N_u N_v g_0 + L_0)(1 + o_0) + \max_{i=1}^{P_0} 4(P_i - 1)(N_u N_v g_i + L_i)(1 + o_i)) \quad (2)$$

where the first expression corresponds to inter-SMP communication and the second corresponds to intra-SMP communication. Where it is assumed that P_i , L_i , o_i , and g_i are the LogP parameters for i th SMP and P_0 , L_0 , o_0 , and g_0 are the LogP parameters corresponding to communications between SMPs.

5 Results and Discussion

Data sets of size $N = 32, 64, 128, 256$, and 512 were simulated and reconstructed using both systems described above. The number of processors varied from one to the maximum available, 18. Each run was repeated three times to average any deviations and to help identify any outliers. The time to complete the execution was measured using the UNIX time command. While time is precise to the nearest tenth of a second, that is sufficient precision when total times ranges from several minutes to several hours. Speedups were modeled for each value of N based on the measured LogSMP parameters.

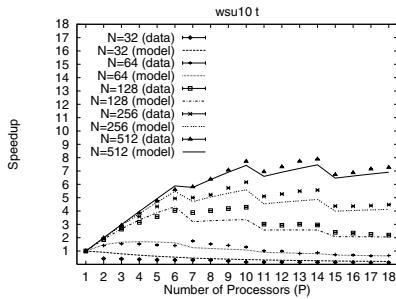


Fig. 2. 10 Mbps ethernet

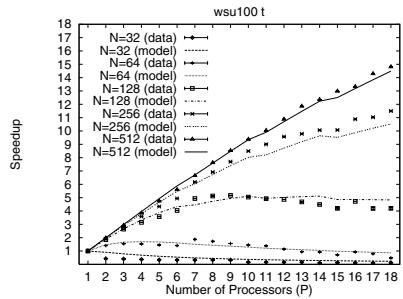


Fig. 3. 100 Mbps ethernet

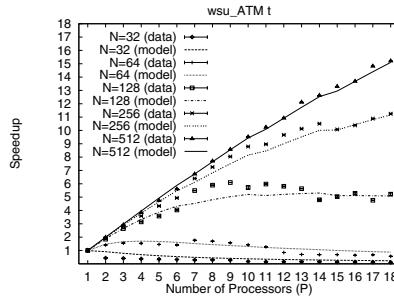


Fig. 4. 155 Mbps ATM

The cluster of SMPs was connected via 10 Mbps ethernet, 100 Mbps ethernet, and then ATM. For each case the reconstruction algorithm was timed and speedup as a function of the number of processors was computed. The speedups

corresponding to the model are plotted along with the empirical data in Fig. 6–8. Recall that the first SMP has six processors and the other three each have four processors. Using 10 Mbps ethernet, a maximal speedup of 7.89 was achieved for a problem size of 512 and using 14 processors. Using 100 Mbps ethernet, a speedup of 14.8 was achieved for a problem size of 512 and using 18 processors. Using ATM, a speedup of 15.2 was achieved for a problem size of 512 and using 18 processors.

The speedup is nearly linear when $P < 6$, since $g \approx 1$. As P increases to force inter-SMP communication, a piecewise continuous behavior is observed and well modeled. By properly accounting for the parameters for each communications channel, an accurate prediction of speedup was obtained. We found the LogSMP model can take into account differences in communication costs and can be easily used to model performance when using clusters of SMPs. This methodology can be applied to similar problems involving large multi-dimensional data sets.

6 Acknowledgements

This research was supported in part by NIH grant AR-43183, NSF grants MIP-9309489, EIA-9729828, US Army Contract DAEA 32-93D004, and Ford Motor Company grants 96-136R and 96-628R.

References

- 1 David A. Reimann, Sean M. Hames, Michael J. Flynn, and David P. Fyhrie. A cone beam computed tomography system for true 3D imaging of specimens. *Applied Radiation and Isotopes*, 48(10–12):1433–1436, October–December 1997.
- 2 David A. Reimann, Vipin Chaudhary, Michael J. Flynn, and Ishwar K. Sethi. Parallel implementation of cone beam tomography. In A. Bojanczyk, editor, *Proceedings of the 1996 International Conference on Parallel Processing*, volume II, pages 170–173, Bloomingdale, Illinois, 12–16 August 1996.
- 3 David A. Reimann, Vipin Chaudhary, Michael J. Flynn, and Ishwar K. Sethi. Cone beam tomography using MPI on heterogeneous workstation clusters. In *Proceedings, Second MPI Developer’s Conference*, pages 142–148, University of Notre Dame, Notre Dame, Indiana, 1–2 July 1996. IEEE Computer Society Press.
- 4 L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone-beam algorithm. *Journal of the Optical Society of America A*, 1:612–19, June 1984.
- 5 David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, California, May 1993.

Session II-A

Compilers and Tools
Chair: Manoj Franklin
University of Maryland

A Fission Technique Enabling Parallelization of Imperfectly Nested Loops

Jialin Ju¹ and Vipin Chaudhary²

¹ Pacific Northwest National Laboratory, Richland WA 99352, USA

Jialin.Ju@pnl.gov,

² Parallel and Distributed Computing Laboratory,
Wayne State University, Detroit MI 48202, USA

vipin@eng.wayne.edu

Abstract. This paper addresses the issue of parallelizing imperfectly nested loops. Current parallelizing compilers or transformations would either only parallelize the inner-most loop (which is more like vectorization than parallelization), or not parallelize the loops at all. We present an approach that transforms an imperfectly nested loop into at most three fully parallel perfectly nested loops. The transformed loops can be parallelized by any parallelizing compiler. The advantage of our technique is the simplicity of the transformed loops and low synchronization overhead. The feasibility of this approach was tested using several types of loops including those from the EISPACK math library and from LINPACK benchmark on different multi-processor platforms and performance was compared with Sun's MP C and Cray's autotasking. The results show that our method is very effective.

1 Introduction

Most scientific/engineering problems spend a majority of their computing time in loops. Loops are classified either as perfectly nested or imperfectly nested based on their structure. Loops can also be classified based on the patterns of the dependences. One is *uniform* dependence and the other *non-uniform* dependence [1]. The dependences are uniform when the dependence vectors can be expressed by *distance vectors*. Otherwise, the dependences are non-uniform. The characteristic of non-uniform dependences is that the array subscripts are coupled (The subscripts are linear combinations of indices). Many standard techniques exist dealing with uniform dependences, such as *loop interchange*, *reversal*, *wavefront*, *tiling*, etc. The research in non-uniform dependence has been limited and most of these techniques assume a perfectly nested loop model. Current parallelizing compilers can't handle non-uniform dependence loops effectively, either leaving them to run sequentially, or giving poor performance even after parallelizing them. An empirical study [2] showed that sixty-six percent of the array references have linear or partially linear subscript expressions. It also showed that approximately forty-five percent of the reference pairs with linear or partially linear subscript expressions have *coupled subscripts*. Loops with coupled subscripts may lead to non-uniform dependences.

This paper focuses on parallelizing imperfectly nested loops, which is the extension of our previous work in [1] (which handles perfectly nested loops). The technique proposed in this paper can handle both uniform and non-uniform dependence structures.

We use the concept of Complete Dependence Convex Hull, Unique Head and Tail Sets as described in [1]. This approach transforms an imperfectly nested loop into at most three fully parallel perfectly nested loops. The transformed loops can be parallelized by any parallelizing compiler. The advantage of our technique is the simplicity of the transformed loops. This method is extremely effective in parallelizing loops with non-uniform dependences.

Most of the techniques proposed for parallelizing loops with non-uniform dependences are based on Dependence Convex Hull theory. These can be classified into two categories: uniform partitioning and non-uniform partitioning. Uniform partitioning includes the work by Tzen and Ni[3], Chen and Yew[4], and Punyamurtula and Chaudhary[5]. Only Chen and Yew's scheme can handle imperfectly nested loop. Large synchronization overhead is the common drawback for these techniques. Non-uniform partitioning includes technique proposed by Zaafrani and Ito[6], and Ju and Chaudhary [1]. Zaafrani and Ito's technique has relatively large sequential region which is a bottleneck for performance. Neither technique can handle imperfectly nested loops. An integer programming approach has also been suggested by Kelly and Pugh [7], which is implemented in the Omega project. Recently Lim and Lam proposed an algorithm [8] which handles loops with non-uniform dependences. The limitation of their work is that their algorithm uses predefined scheduling. Sass and Mutka [9] proposed a technique to convert imperfectly nested loops into perfectly nested loops and then applying unimodular transformations. Not all imperfectly nested loops can be converted into perfectly nested loops. When there are recurrence, induction variables, multiple DO's, or complex data dependences, the transformation would fail. Our technique, on the other hand, generates several fully parallelizable loops, leaving the freedom to the compiler writer to choose the suitable assignment scheme. The transformed code is simple and has less synchronization overhead.

The rest of this paper is organized as follows. Section 2 presents the unique sets oriented parallelization of imperfectly nested loops. Section 3 shows experimental results. Finally, we conclude in section 4.

2 Parallelization of Imperfectly Nested Loops

To avoid cumbersome lengthy expressions, we will illustrate our technique using doubly nested loops. We show how to find unique sets and how to partition the iteration space.

2.1 Finding Unique sets

```
do i = L1, U1
    A[a11 * i + c11, a12 * i + c12] = ...
    do j = L2, U2
        ... = A[a21 * i + b21 * j + c21, a22 * i + b22 * j + c22]
    enddo
enddo
```

The loop nest shown above is the doubly nested loop model we use. The lower and upper bounds of the loop can be unknowns. In real programs, the arrays may be

referenced in different locations. The analyses procedures are the same. Multiple arrays may also be referenced. In such cases, the union of the unique sets would be used. Partitioning would depend on the relationships between the unique sets.

The system of Diophantine equations and the system of linear inequalities are

$$\begin{cases} a_{11} * i_1 + c_{11} = a_{21} * i_2 + b_{21} * j_2 + c_{21} \\ a_{12} * i_1 + c_{12} = a_{22} * i_2 + b_{22} * j_2 + c_{22} \end{cases} \quad (1)$$

$$\begin{cases} L_1 \leq i_1 \leq U_1 \\ j_1 = L_2 - 1 \\ L_1 \leq i_2 \leq U_1 \\ L_2 \leq j_2 \leq U_2 \end{cases} \quad (2)$$

Now we solve for i_2 and j_2 by setting $i_1 = x$ and $j_1 = y$. The solution space \mathbf{S} is the set of points (x, y) satisfying the solution given above. Now the set of inequalities can be written as (3).

$$\begin{cases} L_1 \leq x \leq U_1 \\ y = L_2 - 1 \\ L_1 \leq \alpha_{11}x + \gamma_{11} \leq U_1 \\ L_2 \leq \alpha_{12}x + \gamma_{12} \leq U_2 \end{cases} \quad (3)$$

$$\begin{cases} L_1 \leq \alpha_{21}x + \beta_{21}y + \gamma_{21} \leq U_1 \\ L_1 \leq \alpha_{22}x + \beta_{22}y + \gamma_{22} \leq U_1 \\ L_1 \leq x \leq U_1 \\ L_2 \leq y \leq U_2 \end{cases} \quad (4)$$

where $\alpha_{11} = (a_{11}b_{22} - a_{12}b_{21})/(a_{21}b_{22} - a_{22}b_{21})$, $\gamma_{11} = (b_{22}c_{11} + b_{21}c_{22} - b_{22}c_{21} - b_{21}c_{12})/(a_{21}b_{22} - a_{22}b_{21})$, $\alpha_{12} = (a_{21}a_{12} - a_{11}b_{22})/(a_{21}b_{22} - a_{22}b_{21})$, $\gamma_{12} = (a_{21}c_{12} + a_{22}c_{21} - a_{21}c_{22} - a_{22}c_{11})/(a_{21}b_{22} - a_{22}b_{21})$.

We denote the region defined by the above set of inequalities as **DCH1**. We can further calculate the dependence vectors to be as $d_i(x_1, y_1) = i_2 - i_1 = (\alpha_{11} - 1)x_1 + \gamma_{11}$ and $d_j(x_1, y_1) = j_2 - j_1 = \alpha_{12}x_1 + \gamma_{12} - L_2 + 1$.

Another approach is to solve for i_1 and j_1 by setting $i_2 = x_2$ and $j_2 = y_2$. The solution space \mathbf{S} is the set of points (x, y) satisfying the solution given above. Now the set of inequalities can be written as (4), where $\alpha_{21} = a_{21}/a_{11}$, $\beta_{21} = b_{21}/a_{11}$, $\gamma_{21} = (c_{21} - c_{11})/a_{11}$, $\alpha_{22} = a_{22}/a_{12}$, $\beta_{22} = b_{22}/a_{12}$, $\gamma_{22} = (c_{22} - c_{12})/a_{12}$. We denote the region defined by the above inequalities as **DCH2**. The dependence vectors are $d_i(x_2, y_2) = i_2 - i_1 = (1 - \alpha_{21})x_2 - \beta_{21}y_2 - \gamma_{21}$, $d_i(x_2, y_2) = i_2 - i_1 = (1 - \alpha_{22})x_2 - \beta_{22}y_2 - \gamma_{22}$, and $d_j(x_2, y_2) = j_2 - j_1 = y_2 - L_2 + 1$.

Unique Sets [1] can be derived with the two sets of solution given above. For lack of space, please refer to [10] for details.

2.2 Unique Sets Oriented Partitioning

Sometimes the loop nest may contain only one kind of dependence. If that were the case, the partitioning process would be much simplified. In addition, we would also get very simple parallelized code. Hence, we categorize the loop nests into three cases, one in which there are only flow dependences, another in which there are only anti-dependences, and finally the case in which there are both flow and anti dependences.

The following theorems are developed to distinguish the three cases. Proofs are omitted for lack of space. Please refer to [10].

Theorem 1. *If $d_i(x_1, y_1) = 0$ does not pass through DCH1, then $d_i(x_2, y_2) = 0$ does not pass through DCH2. If $d_i(x_1, y_1) = 0$ passes through DCH1, then $d_i(x_2, y_2) = 0$ must pass through DCH2.*

Theorem 2. If $d_i(x_1, y_1) = 0$ does not pass through any DCH, then there is only one kind of dependence, either flow or anti dependence,

1. if $d_i(x_1, y_1) > 0$ in DCH1,
 - (a) DCH1 is flow dependence unique tail set.
 - (b) DCH2 is flow dependence unique head set.
2. if $d_i(x_1, y_1) < 0$ in DCH1,
 - (a) DCH1 is anti dependence unique head set.
 - (b) DCH2 is anti dependence unique tail set.

Theorem 3. When $d_i(x_1, y_1) = 0$ passes through DCH1, then

1. DCH1 is the union of the flow dependence unique tail set and the anti-dependence unique head set.
2. DCH2 is the union of the flow dependence unique head set and the anti-dependence unique tail set.

Case 1. There is only one kind of dependence which is flow dependence.

DCH1 is the flow dependence unique tail set and DCH2 is the flow dependence unique head set, as shown in Figure 1. Clearly the flow dependence unique head set should be executed after flow dependence unique tail set. Therefore, the doubly imperfectly nested loop in the beginning of this section can be transformed to two perfectly nested parallelizable loops as shown in Figure 4.

Case 2. There is only one kind of dependence which is anti-dependence.

In this case, DCH1 is the anti dependence unique head set and DCH2 is the anti dependence unique tail set, as shown in figure 2. Clearly the anti dependence unique head set should be run after anti dependence unique tail set. Therefore, the doubly imperfectly nested loop in the beginning of this section can be transformed to two perfectly nested parallelizable loops as shown in Figure 5.

Case 3. There are both flow and anti dependences.

In this case, we can divide the DCH inside the iteration space (in this program model, it is DCH2) and run in the following order: *anti dependence unique tail set* \rightarrow DCH1 \rightarrow *flow dependence unique head set*. $d_i(x_2, y_2) = 0$ (There are two $d_i(x_2, y_2)$; the one which leads to a simpler partition can be used) will be used as the dividing line. For the purpose of illustration, we use the first $d_i(x_2, y_2)$ which is $(1 - \alpha_{21})x_2 - \beta_{21}y_2 - \gamma_{21}$. The converted parallel code of our doubly imperfectly nested loop is shown in Figure 6 (Note that it has three perfectly nested parallel loops).

For general nested loops (have nesting more than two), the same procedure applies. Again, there are three cases to consider. If the anti-dependence unique tail set is empty, then there is only one kind of dependence (which is flow dependence) and DCH2 is the same as the flow dependence unique head set. If the flow dependence unique tail set is empty, then there are only anti-dependences and DCH2. For these two cases, we just make two loops with each loop containing one statement as we did in case 1 and case 2 for two dimensional loops. The two loops are fully parallelizable. If neither anti-dependence unique tail set nor flow dependence unique head set is empty, it must be case three that there are both flow and anti dependences. This case is more complicated than the other two. We can run in the order of *anti-dependence unique tail set* \rightarrow DCH1 \rightarrow *flow dependence unique head set*.

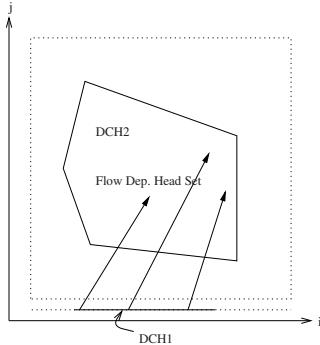


Fig. 1. Case 1: There are only flow dependences

```
doparallel i = L1, U1
  A[a11 * i + c11, a12 * i + c12] = ...
enddo
doparallel i = L1, U1
  doparallel j = L2, U2
    ... = A[a21 * i + b21 * j + c21, a22 * i + b22 * j + c22]
  enddo
enddo
```

Fig. 4. Transformed loop of Case 1

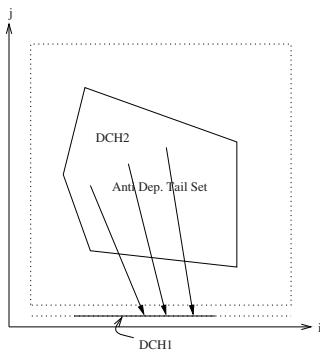


Fig. 2. Case 2: There are only anti dependences

```
doparallel i = L1, U1
  doparallel j = L2, U2
    ... = A[a21 * i + b21 * j + c21, a22 * i + b22 * j + c22]
  enddo
enddo
doparallel i = L1, U1
  A[a11 * i + c11, a12 * i + c12] = ...
enddo
```

Fig. 5. Transformed Loop of Case 2

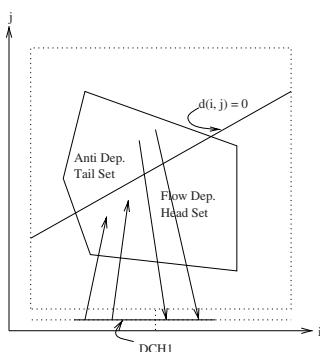


Fig. 3. Case 3: There are both flow and anti Dependences

```
doparallel i = L1, U1
  doparallel j = L2, U2
    if ((1 - α21)i - β21j - γ21) ≤ 0   (< 0 if
di(x2, y2) = 0
      belongs to anti dependence tail set)
    ... = A[a21*i+b21*j+c21, a22*i+b22*j+c22]
    endif
  enddo
enddo
doparallel i = L1, U1
  A[a11 * i + c11, a12 * i + c12] = ...
enddo
doparallel i = L1, U1
  doparallel j = L2, U2
    if ((1 - α21)i - β21j - γ21) > 0   (≥ 0 if
di(x2, y2) = 0
      belongs to flow dependence head set)
    ... = A[a21*i+b21*j+c21, a22*i+b22*j+c22]
    endif
  enddo
enddo
```

Fig. 6. Transformed Loop of Case 3

3 Experimental Results

```

do i = 1, SIZE           do i = 1, SIZE
  A(2i + 4, i+5) = ...
  do j = 1, SIZE         ... = A(i, N * i + b)
    ... = A(i + 2j + 2, i + j)   do j = i+1, SIZE
    enddo                   A(i, j) = ...
  enddo                   enddo
Loop 1                  enddo

do i = 1, SIZE           do i = 1, SIZE
  do j = 1, SIZE         A(i, j) = 1.0 / A(i, i)
    do k = i, SIZE       do j = i+1, SIZE
      ... = A(N-j, k)   A(j, i) = A(j,i) * A(i, i)
    enddo                 do k = i+1, SIZE
    A(N-j+1, i) = ...   A(j,k) = A(j,k) - A(j,i)
    enddo                   * A(i,k)
  enddo                   enddo
enddo                     enddo
Loop 2

do i = 1, SIZE           do i = 1, SIZE
  do j = 1, SIZE         A(i, j) = 1.0 / A(i, i)
    do k = i, SIZE       do j = i+1, SIZE
      ... = A(N-j, k)   A(j, i) = A(j,i) * A(i, i)
    enddo                 do k = i+1, SIZE
    A(N-j+1, i) = ...   A(j,k) = A(j,k) - A(j,i)
    enddo                   * A(i,k)
  enddo                   enddo
enddo                     enddo
Loop 3                  enddo

do i = 1, SIZE           do i = 1, SIZE
  do j = 1, SIZE         A(i, j) = 1.0 / A(i, i)
    do k = i, SIZE       do j = i+1, SIZE
      ... = A(N-j, k)   A(j, i) = A(j,i) * A(i, i)
    enddo                 do k = i+1, SIZE
    A(N-j+1, i) = ...   A(j,k) = A(j,k) - A(j,i)
    enddo                   * A(i,k)
  enddo                   enddo
enddo                     enddo
Loop 4

```

We present the results for four loops. Loop 1 is a two dimensional loop with non-uniform dependences. The loop size (*SIZE*) used in the experiments is 1500. Loop 2 is taken from *Linpack* benchmark and Loop 3 is taken from *qzhes,f* of *Eispack*. Many statements have been removed from the original loops in order to simplify the loop. Only one pair of array is considered. The second array subscript of the first array reference in Loop 2 was a function call in the original array. Here we assume that the result of the function returns $N * i + b$ and we assume $N = 2$ and $b = 3$. The loop size (*SIZE*) used in the experiments is 1500 for Loop 2 and 500 for Loop 3. The last loop is the standard Gauss Elimination code. The loop size (*SIZE*) used in the experiments is 1500.

We present the results on two multi-processor architectures. One is a Ultra Enterprise 4000 with four 248MHz Ultra Sparc processors and 512 MB memory. The other is a Cray J916 with 16 processors and 4 GBytes of RAM.

Sun MP C is an extended ANSI C compiler that can compile code to run on SPARC shared memory multiprocessor machines. The MP Compiler is able to perform extensive automatic restructuring of user code. These automatic transformations include: loop interchange, loop fusion, loop distribution, and software pipelining.

Figure 7 (a) - (d) shows the performance for Loop 1 to Loop 4 on Sun MP. Loop1 has both flow and anti dependences. Our transformation yields three fully parallel perfectly nested loops. The program synchronizes only three times. This is a major advantage of our technique. Loop 2 has uniform dependences. The Sun compiler performs well with this loop and our technique is marginally better. Loop 3 is a loop with non-uniform dependences. Our method obtained a super linear speedup of 6.4 with 4 threads running on 4 processors. Even when using one thread, our method ran faster than the sequential version of the program. This is because our transformation makes the code amenable to instruction level parallelism and the native compiler can exploit this. For this loop, our

transformed loop synchronizes twice. Loop 4 gave a speedup of 3.80 with 4 processors and outperformed the Sun compiler which had a speedup of 3.15 with 4 processors.

On Cray J916, we use the *Autotasking Expert System*(atexpert) to analyze the program. Figure 7 (e) - (h) show the speedup comparison of our technique and Cray's autotasking for Loop 1 to Loop 4. Cray's autotasking could not parallelize Loop 1 at all. The transformed loop by our technique shows excellent scaling in speedup with a maximum of 12.28 using sixteen processors. For Loop 2, the speedup of our method and that of Cray's Autotasking are very similar. The speedup is rather low due to the memory organization of J916. For Loop 3, Our technique shows "near" linear speedup. Our technique only synchronizes twice for this program. Cray's native compiler failed to parallelize this loop. For the last loop, Loop 4, our method scales very well and achieved a speedup of 9.41 for 16 processors whereas Cray's autotasking achieved speedup of 4.64 for the same number of processors.

It is clear from the results that our transformation generates code that is parallelized by parallelizable compilers for RISC as well as vector supercomputers. The speedup also scales well. The transformation is especially effective if the loop has non-uniform dependences. The differences of the performance of Loop 1 and Loop 3 when compiled with native compilers on Sun MP and Cray are because the Sun compiler parallelized only the inner-most loop, while Cray run the loops sequentially.

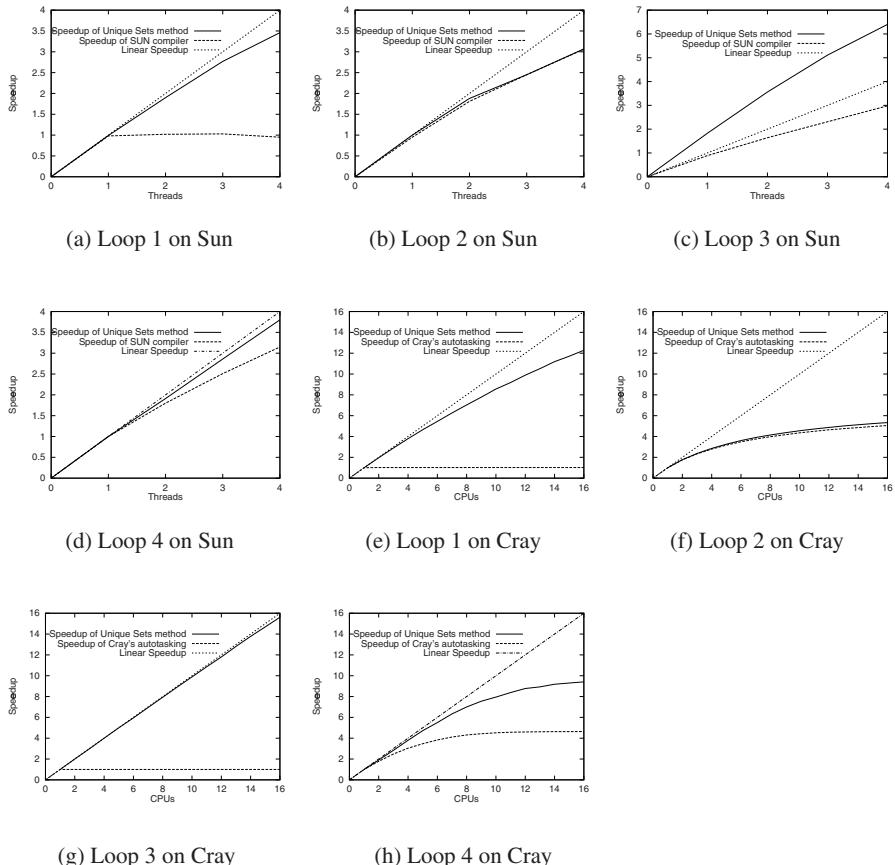
4 Conclusion

This paper presented a transformation technique that splits an imperfectly nested loop into several fully parallelizable loops. The proposed technique is mathematically sound and has at most three synchronizations in the transformed code. Another major advantage of this technique is that the transformed code has a very simple structure which is shown to be amenable to instruction level optimizations. The proposed technique handles imperfectly nested loops with uniform as well as non-uniform dependences. The performance gains for non-uniform dependences is especially remarkable (at times achieving super-linear speedups). Experiments with commercial compilers for different architectures on loops from real programs confirm the efficacy of this approach.

This research was supported in part by NSF grants MIP-9309489, EIA-9729828, US Army Contract DAEA 32-93D004, and Ford Motor Company grants 96-136R and 96-628R.

References

1. Ju, J., Chaudhary, V.: Unique sets oriented parallelization of loops with non-uniform dependences. *The Computer Journal*, Vol.40, **6** (1997) 322–339
2. Shen, Z., Li, Z., Yew, P.C.: An empirical study on array subscripts and data dependencies. *Proc. 1989 Int. Conf. on Parallel Processing. II* 145–152
3. Tzen, T.H., Ni, L.M.: Dependence uniformization: A loop parallelization technique. *IEEE trans. Parallel and Distrib. Syst.* **4** (1993) 547–558
4. Chen, D.L., Yew, P.C.: On effective execution of nonuniform doacross loops. *IEEE trans. Parallel and Distrib. Syst.* **7** (1996) 463–476

**Fig. 7. Performance**

5. Punyamurtula, S., Chaudhary, V., Ju, J., Roy, S.: Compile time partitioning of nested loop iteration spaces with non-uniform dependences. *J. Parallel Algor. and App.* **12** (1996) 113–141
6. Zaafrani, A., Ito, M.R.: Parallel region execution of loops with irregular dependences. Proc. 1994 Int. Conf. Parallel Processing. II 11–19
7. Kelly, W., Pugh, W.: Minimizing communication while preserving parallelism. Int. Conf. Supercomputing (1996)
8. Lim, A.W., Lam, M.S.: Maximizing parallelism and minimizing synchronization with affine transforms. 24th Ann. ACM SIGPLAN-SIGACT Symp. Prin. Prog. Lang. Paris (1997)
9. Sass, R., Mutka, M.: Enabling unimodular transformations. Proc. Supercomputing '94 (1994) 753–762
10. Ju, J.: Automatic Parallelization of Non-Uniform Loops. Ph.D. thesis. Wayne State University (1998)

A Novel "Bi-directional Execution" Approach to Debugging Distributed Programs

R. Mall

School of Computing
Curtin University of Technology,
GPO Box U 1987, Perth 6845, Australia
E-mail: rajib@cs.curtin.edu.au

Abstract. Conventional debuggers do not allow users to go back and examine the program states at statements which have already been executed. In case the user wants to examine the program state at a statement which was executed sometime back, he is forced to restart the entire debugging process. To overcome this problem, we examine the issue of bidirectional execution of programs. To this end, we introduce the concepts of *inverse of a statement* and *inverse of a program*. We describe our implementation of a debugger which can execute distributed programs in either forward or backward direction depending upon an option dynamically set by the user.

1 Introduction

Debugging tools play a vital role in any software development effort. Appropriate debugging tools are especially indispensable to the development of distributed programs. Not only are distributed applications inherently complex, but they possess several other intrinsic properties which make them difficult to debug. Some of these features of distributed programs are:

- Inherent non-determinism of programs due to concurrent executions at multiple sites.
- Lack of global states.
- Multiple threads of control.

While debugging, often one realizes that an error had probably occurred at some earlier program statements. But, conventional debuggers have no way to restore execution state to a previous point. It is especially frustrating to find that after a long and tedious debugging session, we have over-stepped the defective statement. The only possibility in such a situation is to restart the entire debugging process. Even restarting the program execution often does not help due to the inherent nondeterminism of the distributed programs. Therefore, for distributed programs it would be better if the debugger could reverse-execute statements to reach the desired point of execution quickly.

This brings us to an interesting question, "can we compute the inverse of different statements in a program and go back to a state that existed before

some statement was executed?" We have examined this question in the context of C programs[1]. We extend this concept to handle reverse execution of distributed programs. We have used the ideas developed in this paper to design and implement a debugger called BDD(Bi-directional Distributed Debugger). BDD allows users to dynamically set an option called "direction of execution" to indicate execution of a program either in forward or backward direction and the subsequent execution proceeds based on the value set for this option.

1.1 Related Work

The issue of reversibility of computation has been addressed by researchers at various levels. Issues involved in reverse computations at the level of logic gates and the feasibility of reversible Turing machines have been discussed by Bennett et al [7]. An interesting work exploring reverse execution of machine language programs is reported by Cezzar [5]. Cezzar focussed on designing an instruction set for reversible execution. He also proposed a mechanism to save the overwritten values on an internal stack associated with instruction decodings. The values stacked during forward execution are later retrieved during reverse execution to restore the corresponding variables to their prior values.

Briggs [8] examined reverse execution of high-level languages. He describes the design of a system to control the operation of an electronic cricket score-board. The main feature implemented in his system is the ability of the cricket score board operator to 'undo' operations that he had already performed, in order to correct inadvertently errors made. Reverse execution of the cricket score board program is performed to achieve the undo operations. Reverse execution of assignment statements, conditional statements and iterative statements of Pascal programs have been discussed in this context. The primitive operation necessary to allow an operator to change events is 'undo' — return the program to the state it was in prior to the event taking place. Many existing systems provide an undo operation — by Leeman [9] lists many of them. However, none of these work has investigated reverse execution of programs in the context of program debugging. We have recently reported our preliminary results of reverse execution of programs in [1]. We further extend the concept of reverse execution to distributed programs.

The rest of this paper is organized as follows. In section 2, the computation of inverse for various categories of program statements is examined. In section 3, we discuss the overall schema of our debugger BDD. section 4 concludes this paper.

2 Inverse of a Program

We define the inverse of a statement S to be another statement S' which when executed restores the state of the program to that which existed before S was executed. By extending this definition, we can define the inverse of a program P to be another program P' which when executed restores every execution step of

the original program. Thus, if we can define the inverse of a program, we would have an elegant means of reverse execution.

A naive way of achieving reverse execution of a program is to keep track of the entire execution trace, i.e. recording the *program state* after execution of each statement. The term *program state* as used here means the values of all the variables active at a point in a program. The volume of the information one needs to store in such a naive approach would be enormous for large programs. However, if we can roll back to the previous states by executing the inverse statements, the volume of the execution trace to be recorded can be reduced several folds. We will see later that we cannot entirely do away with maintaining a trace file, since some types of statements would need recording execution trace for reverse execution. An important objective of our approach is to minimize the size of the trace information to be maintained.

As we have already mentioned, we shall be considering reverse execution of only C programs augmented with message-passing library calls. In order to compute the inverse of a statement, we classify C programming constructs [3] into the following types :

1. Sequence statements, e.g. assignment statements.
2. Selection statements, e.g. "if-else" statement, "case" statement, etc.
3. Iterative statements, e.g. "while" statement, "for" statement, etc.
4. Unstructured statements, e.g. "goto" statement, "break" statement, etc.
5. Function call statements.

The issues relating to computation of inverses of these different categories of program statements have been discussed in [1]. To support distributed execution, we should be able to handle reverse execution of the message passing statements. In the next subsection we describe how we extend this technique to handle distributed programs.

2.1 Handling Distributed Programs

A distributed program consists of a number of processes executing at different sites. The processes may interact with each other through message passing. The salient features through which reverse execution of distributed programs are supported are the following:

- The execution trace for each process is collected in a separate file and is maintained at the local sites (see Fig. 2).
- The user is provided with a control panel in an independent window to set the direction of execution (see fig. 1).
- The state of each process and the relevant program code is displayed in a different window in the control panel.

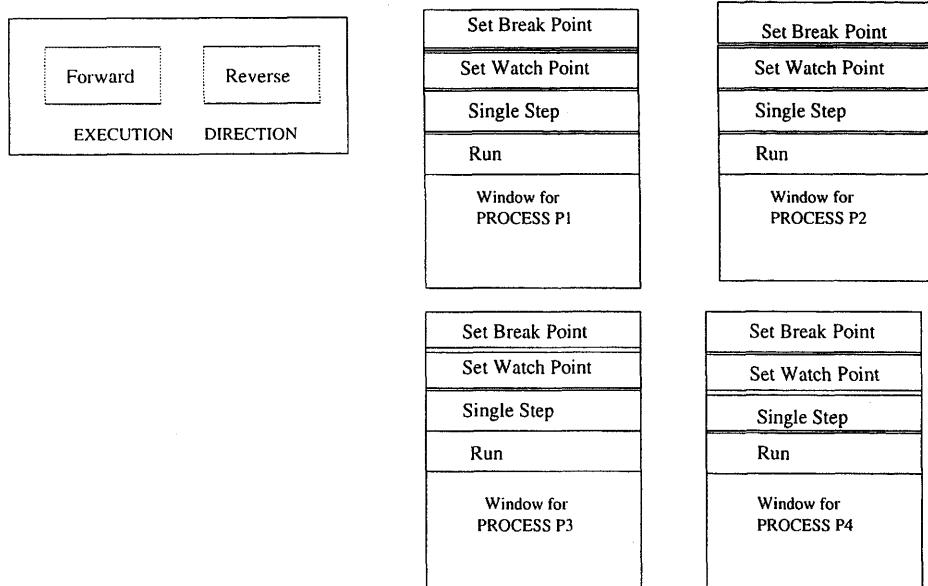


Fig. 1. Schematic Representation of Distributed Execution

- The windows for different processes are decorated with menu options for single-stepping, setting break points, setting watch points, and running the process. (see Fig. 1)
- It may not always be possible to single step through a process in the forward direction, especially if a process is waiting at a synchronization point. For example, consider the case where processes i and j synchronize at some point. In such a situation, any attempt to forward execute process i past the synchronization point without satisfying the synchronization condition generates a message: "waiting for a message from process j", where j is the number of the processes from which the message is expected. Therefore, one would have to execute process j past the synchronization point before process i can be executed forward.

However, for reverse execution, trying to reverse execute one process past the synchronization point reverse executes both the processes involved in synchronization. Note that this may lead to a cascade of reverse execution steps in different processes.

The inverse statement of a message passing statement depends on the semantics of message passing. The commonly used synchronization semantics are the following:

- **Partially Blocking :** In this case, the sending process sends the message and then proceeds with other computation. But the receiving process blocks until it receives the required message from the sending process.

- **Fully Blocking :** In this case, both sending and receiving processes block. That is, the sending process and the receiving process block until both are ready for communication.

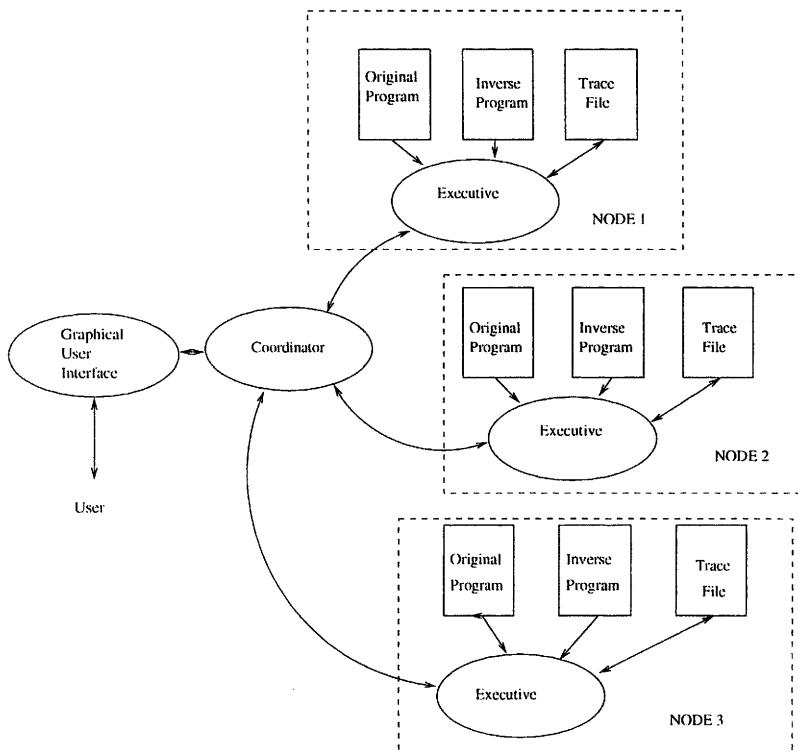


Fig. 2. High-level Schema of BDD

In the rest of the paper we will assume a partially blocking semantics of message communication. However, our results can be easily extended to other types of synchronization semantics. When a `recv_msg` is executed in the forward direction, the recent values of the variables that are overwritten during the message receive operation are recorded in the trace file. During reverse execution of the `recv_msg` the trace file is read to reset the variable values. For `send_msg`, no special handling is required either during the forward or reverse execution. During reverse execution of the `send_msg`, a "null" instruction is executed.

3 Overview of Design of BDD

Using BDD, the user can forward execute his program, and at any time during forward execution he can start reverse execution of his program, he can set break points, check program state, etc. An overview of our design of BDD is schematically shown in *Figure 2*. Copies of the debugger executives reside at every node participating in the debugging process. A coordinator process resides at the node where the BDD is invoked. The coordinator interacts with the executives at the local nodes to carry out the debugging process. At every node local copies of the code of the program and corresponding to the process executing at the node and the corresponding inverse program exist. Local copies of the trace files are created as the debugging session proceeds (see Fig. 2). In fig. 2, the inverse Program contains the inverse statements of the original program. The trace file contains the information needed to reverse execute statements for which pure inverses do not exist are kept in the trace file. Several other information such as symbol tables corresponding to the function calls, goto statements denoting repeated execution of loop constructs, and information relating to message passing are also recorded in the trace file.

- **GUI** - The GUI reads the user command such as the direction of execution, commands for single stepping, commands to inspect variable values, the command to execute different processes, etc. The relevant information for the user is displayed in proper format.
- **Inverse Program Generator** - This module compiles the source code to generate the inverse program.
- **Executive** - This module controls the direction of program execution. *Figure 3* shows the functional components of the executive module. Some of the important information maintained by an executive are :

Program State - The program state, at any point during execution is the value of all variables active at that point.

Statement Pointer - The statement pointer maintains the statement number of the next instruction to be executed. The exact implementation of this pointer is a bit more complicated since it should point to the next instruction in both the forward and the reverse programs and the actual instruction to be fetched depends on the "direction of execution" option set.

Trace Pointer - The trace pointer points to the relevant location in the trace file. When some information is to be recorded, it is written in the trace file at the position pointed to by the trace pointer. When we need to read information from the trace file, the required position is the instruction preceding the one pointed to by the trace pointer.

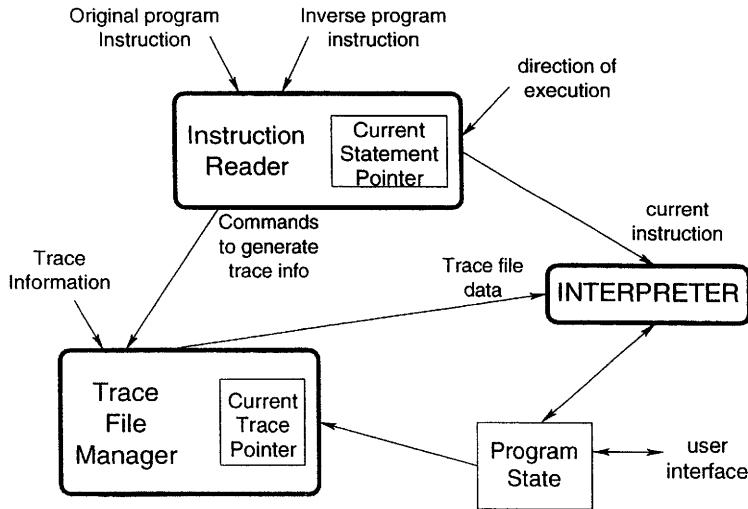


Fig. 3. Schematic design of the Executive

The main role of the different functional components of the executive are as outlined below:

Instruction Reader - Depending on the direction of execution, the next statement to be executed is selected. As shown in *Figure 3* the statement pointer keeps track of the next statement to be executed. This module instructs the trace file manager to write the information to be remembered into the trace file.

Trace File Manager - This module writes the program state into the trace file at the appropriate point. It also retrieves the relevant information from the trace file whenever required during reverse execution.

Interpreter - The selected statement is executed by the interpreter sub-module which also maintains the program execution state.

- **Coordinator:** The coordinator interfaces with the GUI and depending on the direction of execution set by the user and depending upon the other debugging commands selected by the user, the coordinator module coordinates the actions of the appropriate executive modules to perform the debugging command issued by the user.

4 Conclusion

We have examined the problem of reverse execution of distributed programs as means to facilitate program debugging. We have shown that meaningful inverse for only a subset of program statements can be defined. For other statements, we must maintain the value transformations achieved by those statements in

order to do a reverse transformation during backward execution. We have designed and implemented a debugging tool which can execute a program in either the forward or reverse direction in a Unix Network environment. In our implementation for distributed execution, trace files are maintained at different sites. Also, copies of the debugger executives execute at local sites. The GUI of the implemented debugger interacts with the user and a coordinator module which coordinates the actions of different process debugging executives. We find this tool to be very useful during our initial experimentation.

Acknowledgements: The author acknowledges the help of Mr. Bitan Biswas in implementing the debugger BDD.

References

1. Bitan Biswas and R. Mall, "Reverse Execution of Programs," ACM SIGPLAN Notices, 34(4), pp. 61–69, April, 1999.
ACM SIGPLAN Notices,
2. R. Mall, *Fundamentals of Software Engineering*, Prentice Hall of India, New Delhi, 1998.
3. B. Kerninghan and D. Ritchie, *The C Programming Language*, Prentice Hall of India, New Delhi, 1988.
4. John D. Johnson and Gary W. Kenney, *Implementation Issues for a Source Level Symbolic Debugger(Extended Abstract)*, ACM SIGPLAN Notices, 18(8), pp. 149–151, Aug, 1983.
5. R. Cezzar, *The Design of a Processor Architecture Capable of Forward and Reverse Execution*, Proceedings of IEEE SOUTHEASTCON '91, pp. 885–890, Vol. 2, Apr, 1991.
6. B. Biswas, *Reverse Execution of Programs*, Master's Thesis, CSE Department, IIT, Kharagpur, India, December, 1998.
7. C. H. Bennett and R. Landauer, *The Fundamental Physical Limits of Computation*, Sci. American, Vol. 253, July 1985.
8. J. S. Briggs, *Generating Reversible Programs*, Software — Practice and Experience, Vol. 17(7), 439–453, July 1987.
9. G. B. Leeman Jr, *A formal Approach to undo Operations in Programming Languages*, ACM Transactions on Programming Languages and Systems, 8, (1), pp. 50–87, 1986.

Memory-Optimal Evaluation of Expression Trees Involving Large Objects *

Chi-Chung Lam¹, Daniel Cociorva², Gerald Baumgartner¹, and P. Sadayappan¹

¹ Department of Computer and Information Science

The Ohio State University, Columbus, OH 43210

{c1am,gb,saday}@cis.ohio-state.edu

² Department of Physics

The Ohio State University, Columbus, OH 43210

cociorva@pacific.mps.ohio-state.edu

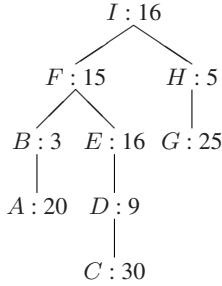
Abstract. The need to evaluate expression trees involving large objects arises in scientific computing applications such as electronic structure calculations. Often, the tree node objects are very large that only a subset of them can fit in memory at a time. This paper addresses the problem of finding an evaluation order of nodes in a given expression tree that uses the least memory. We develop an efficient algorithm that finds an optimal evaluation order in $O(n^2)$ time for an n -node expression tree.

1 Introduction

This paper addresses the problem of finding an evaluation order of the nodes in a given expression tree that minimizes memory usage. The expression tree must be evaluated in some bottom-up order, i.e., the evaluation of a node cannot precede the evaluation of any of its children. The nodes of the expression tree are large data objects whose sizes are given. If the total size of the data objects is so large that they cannot all fit into memory at the same time, space for the data objects has to be allocated and deallocated dynamically. Due to the parent-child dependence relation, a data object cannot be deallocated until its parent node data object has been evaluated. The objective is to minimize the maximum memory usage during the evaluation of the entire expression tree.

This problem arises, for example, in optimizing a class of loop calculations implementing multi-dimensional integrals of the products of several large input arrays that computes the electronic properties of semiconductors and metals [2,3,9]. The multi-dimensional integral can be represented as an expression tree in which the leaf nodes are input arrays, the internal nodes are intermediate arrays, and the root is the final integral. In previous work, we have addressed the problems of 1) finding an expression tree with the minimal number of arithmetic operations and 2) the mapping of the computation on parallel computers to minimize the amount of inter-processor communication [4,5,6]. However, in practice, the input arrays and the intermediate arrays are often so large that they cannot all fit into available memory. There is a need to allocate array space dynamically and to evaluate the arrays in an order that uses the least memory. Solving

* Supported in part by the National Science Foundation under grant DMR-9520319.

**Fig. 1.** An example expression tree

this problem would help the automatic generation of code that computes the electronic properties. We believe that the solution we develop here may have potential applicability to other areas such as database query optimization and data mining.

As an example of the memory usage optimization problem, consider the expression tree shown in Fig. 1. The size of each data object is shown alongside the corresponding node label. Before evaluating a data object, space for it must be allocated. That space can be deallocated only after the evaluation of its parent is complete. There are many allowable evaluation orders of the nodes. One of them is the post-order traversal $\langle A, B, C, D, E, F, G, H, I \rangle$ of the expression tree. It has a maximum memory usage of 45 units. This occurs during the evaluation of H , when F , G , and H are in memory. Other evaluation orders may use more memory or less memory. Finding the optimal order $\langle C, D, G, H, A, B, E, F, I \rangle$, which uses 39 units of memory, is not trivial.

A simpler problem related to the memory usage minimization problem is the register allocation problem in which the sizes of all nodes are unity. It has been addressed in [8,10] and can be solved in $O(n)$ time, where n is the number of nodes in the expression tree. But if the expression tree is replaced by a directed acyclic graph (in which all nodes are still of unit size), the problem becomes NP-complete [11]. The algorithm in [10] for expression trees of unit-sized nodes does not extend directly to expression trees having nodes of different sizes. Appel and Supowit [1] generalized the register allocation problem to higher degree expression trees of arbitrarily-sized nodes. However, the problem they addressed is slightly different from ours in that, in their problem, space for a node is not allocated during its evaluation. Also, they restricted their attention to solutions that evaluate subtrees contiguously, which is sub-optimal in some cases. We are not aware of any existing algorithm to the memory usage optimization problem considered in this paper.

The rest of this paper is organized as follows. In Section 2, we formally define the memory usage optimization problem and make some observations about it. Section 3 presents an efficient algorithm that solves the problem in $O(n^2)$ time for an n -node expression tree. Section 4 provides conclusions. Due to space constraints, the correctness proof of the algorithm is omitted from this paper but can be found in [7].

2 Problem Statement

The problem addressed is the optimization of memory usage in the evaluation of a given expression tree, whose nodes correspond to large data objects of various sizes. Each data object depends on all its children (if any), and thus can be evaluated only after all its children have been evaluated. The goal is to find an evaluation order of the nodes that uses the least amount of memory. Since an evaluation order is also a traversal of the nodes, we will use these two terms interchangeably. Space for data objects is dynamically allocated or deallocated under the following assumptions:

1. Each object is allocated or deallocated in its entirety.
2. Leaf node objects are created or read in as needed.
3. Internal node objects must be allocated before their evaluation begins.
4. Each object must remain in memory until the evaluation of its parent is completed.

We define the problem formally as follows:

Given a tree T and a size $v.\text{size}$ for each node $v \in T$, find a computation of T that uses the least memory, i.e., an ordering $P = \langle v_1, v_2, \dots, v_n \rangle$ of the nodes in T , where n is the number of nodes in T , such that

1. for all v_i, v_j , if v_i is the parent of v_j , then $i > j$; and
2. $\max_{v_i \in P} \{\text{himem}(v_i, P)\}$ is minimized, where

$$\text{himem}(v_i, P) = \text{lomem}(v_{i-1}, P) + v_i.\text{size}$$

$$\text{lomem}(v_i, P) = \begin{cases} \text{himem}(v_i, P) - \sum_{\{\text{child } v_j \text{ of } v_i\}} v_j.\text{size} & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases}$$

Here, $\text{himem}(v_i, P)$ is the memory usage during the evaluation of v_i in the traversal P , and $\text{lomem}(v_i, P)$ is the memory usage upon completion of the same evaluation. In general, we need to allocate space for v_i before its evaluation. After v_i is evaluated, the space allocated to all its children may be released. For instance, consider the post-order traversal $P = \langle A, B, C, D, E, F, G, H, I \rangle$ of the expression tree shown in Fig. 1. During and after the evaluation of A , A is in memory. So, $\text{himem}(A, P) = \text{lomem}(A, P) = A.\text{size} = 20$. To evaluate B , we need to allocate space for B , thus $\text{himem}(B, P) = \text{lomem}(A, P) + B.\text{size} = 23$. After B is obtained, A can be deallocated, giving $\text{lomem}(B, P) = \text{himem}(B, P) - A.\text{size} = 3$. The memory usage for the rest of the nodes is determined similarly and shown in Fig. 2(a).

However, the post-order traversal of the given expression tree is not optimal in memory usage. For this example, none of the traversals that visit all nodes in one subtree before visiting another subtree is optimal. For the given expression tree, there are four such traversals. They are $\langle A, B, C, D, E, F, G, H, I \rangle$, $\langle C, D, E, A, B, F, G, H, I \rangle$, $\langle G, H, A, B, C, D, E, F, I \rangle$, and $\langle G, H, C, D, E, A, B, F, I \rangle$. If we follow the traditional wisdom of visiting the subtree that uses more memory first, we obtain the best of the four traversals, which is $\langle G, H, C, D, E, A, B, F, I \rangle$. Its overall memory usage is 44 units, as shown in Fig. 2(b), and is not optimal. The optimal traversal is $\langle C, D, G, H, A, B, E, F, I \rangle$, which uses 39 units of memory (see Fig. 2(c)). Notice that it ‘jumps’ back and forth between the subtrees. Therefore, any algorithm that only considers traversals that visit subtrees contiguously may not produce an optimal solution.

Node	himem	lomem	Node	himem	lomem	Node	himem	lomem
A	20	20	G	25	25	C	30	30
B	23	3	H	30	5	D	39	9
C	33	33	C	35	35	G	34	34
D	42	12	D	44	14	H	39	14
E	28	19	E	30	21	A	34	34
F	34	15	A	41	41	B	37	17
G	40	40	B	44	24	E	33	24
H	45	20	F	39	20	F	39	20
I	36	16	I	36	16	I	36	16
max	45		max	44		max	39	

(a) Post-order traversal (b) A better traversal (c) The optimal traversal

Fig. 2. Memory usage of three different traversals of the expression tree in Fig. 1

One possible approach to the memory usage optimization problem is to apply dynamic programming on an expression tree as follows. Each traversal can be viewed as going through a sequence of configurations, each configuration being a set of nodes that have been evaluated (which can be represented more compactly as a smaller set of nodes in which none is an ancestor or descendant of another). In other words, the set of nodes in a prefix of a traversal forms a configuration. Common configurations in different traversals form overlapping subproblems. A configuration can be formed in many ways, corresponding to different orderings of the nodes. The optimal way to form a configuration Z containing k nodes can be obtained by minimizing over all valid configurations that are $k - 1$ -subsets of Z . By finding the optimal costs for all configurations in the order of increasing number of nodes, we get an optimal traversal of the expression tree. However, this approach is inefficient in that the number of configurations is exponential in the number of nodes.

The memory usage optimization problem has an interesting property: an expression tree or a subtree may have more than one optimal traversal. For example, for the subtree rooted at F , the traversals $\langle C, D, E, A, B, F \rangle$ and $\langle C, D, A, B, E, F \rangle$ both use the least memory space of 39 units. One might attempt to take two optimal subtree traversals, one from each child of a node X , merge them together optimally, and then append X to form a traversal for X . But, this resulting traversal may not be optimal for X . Continuing with the above example, if we merge together $\langle C, D, E, A, B, F \rangle$ and $\langle G, H \rangle$ (which are optimal for the subtrees rooted at F and H , respectively) and then append I , the best we can get is a sub-optimal traversal $\langle G, H, C, D, E, A, B, F, I \rangle$ that uses 44 units of memory (see Fig. 2(b)). However, the other optimal traversal $\langle C, D, A, B, E, F \rangle$ for the subtree rooted at F can be merged with $\langle G, H \rangle$ to form $\langle C, D, G, H, A, B, E, F, I \rangle$ (with I appended), which is an optimal traversal of the entire expression tree. Thus, locally optimal traversals may not be globally optimal. In the next section, we present an efficient algorithm that finds traversals which are not only locally optimal but also globally optimal.

3 An Efficient Algorithm

We now present an efficient divide-and-conquer algorithm that, given an expression tree whose nodes are large data objects, finds an evaluation order of the tree that minimizes the memory usage. For each node in the expression tree, it computes an optimal traversal for the subtree rooted at that node. The optimal subtree traversal that it computes has a special property: it is not only locally optimal for the subtree, but also globally optimal in the sense that it can be merged together with globally optimal traversals for other subtrees to form an optimal traversal for a larger subtree which is also globally optimal. As we have seen in Section 2, not all locally optimal traversals for a subtree can be used to form an optimal traversal for a larger tree.

The algorithm stores a traversal not as an ordered list of nodes, but as an ordered list of indivisible units called elements. Each element contains an ordered list of nodes with the property that there necessarily exists some globally optimal traversal of the entire tree wherein this sequence appears undivided. Therefore, as we show later, inserting any node in between the nodes of an element does not lower the total memory usage. An element initially contains a single node. But as the algorithm goes up the tree merging traversals together and appending new nodes to them, elements may be appended together to form new elements containing a larger number of nodes. Moreover, the order of indivisible units in a traversal stays invariant, i.e., the indivisible units must appear in the same order in some optimal traversal of the entire expression tree. This means that indivisible units can be treated as a whole and we only need to consider the relative order of indivisible units from different subtrees.

Each element (or indivisible unit) in a traversal is a $(\text{nodelist}, hi, lo)$ triple, where nodelist is an ordered list of nodes, hi is the maximum memory usage during the evaluation of the nodes in nodelist , and lo is the memory usage after those nodes are evaluated. Using the terminology from Section 2, hi is the highest himem among the nodes in nodelist , and lo is the lomem of the last node in nodelist . The algorithm always maintains the elements of a traversal in decreasing hi and increasing lo order, which implies in order of decreasing $hi-lo$ difference.

Fig. 3 shows the algorithm. The input to the algorithm (the **MinMemTraversal** procedure) is an expression tree T , in which each node v has a field $v.size$ denoting the size of its data object. The procedure performs a bottom-up traversal of the tree and, for each node v , computes an optimal traversal $v.seq$ for the subtree rooted at v . The optimal traversal $v.seq$ is obtained by optimally merging together the optimal traversals $u.seq$ from each child u of v , and then appending v . At the end, the procedure returns a concatenation of all the nodelists in $T.root.seq$ as the optimal traversal for the given expression tree. The memory usage of the optimal traversal is $T.root.seq[1].hi$.

The **MergeSeq** procedure merges two given traversals $S1$ and $S2$ optimally and returns the merged result S . $S1$ and $S2$ are subtree traversals of two children nodes of the same parent. The optimal merge is performed in a fashion similar to merge-sort. Elements from $S1$ and $S2$ are scanned sequentially and appended into S in the order of decreasing $hi-lo$ difference. This order guarantees that the indivisible units are arranged to minimize memory usage. Since $S1$ and $S2$ are formed independently, the $hi-lo$ values in the elements from $S1$ and $S2$ must be adjusted before they can be appended to S .

MinMemTraversal (T):

```

foreach node  $v$  in some bottom-up traversal of  $T$ 
   $v.seq = \langle \rangle$  // an empty list
  foreach child  $u$  of  $v$ 
     $v.seq = \text{MergeSeq}(v.seq, u.seq)$ 
  if  $|v.seq| > 0$  then //  $|x|$  is the length of  $x$ 
     $base = v.seq[|v.seq|].lo$ 
  else
     $base = 0$ 
  AppendSeq ( $v.seq, \langle v \rangle, v.size + base, v.size$ )
   $nodelist = \langle \rangle$ 
  for  $i = 1$  to  $|T.root.seq|$ 
     $nodelist = nodelist + T.root.seq[i].nodelist$  // + is the concatenation operator
  return  $nodelist$  // memory usage is  $T.root.seq[1].hi$ 
```

MergeSeq ($S1, S2$):

```

 $S = \langle \rangle$ 
 $i = j = 1$ 
 $base1 = base2 = 0$ 
while  $i \leq |S1|$  or  $j \leq |S2|$ 
  if  $j > |S2|$  or  $(i \leq |S1| \text{ and } S1[i].hi - S1[i].lo > S2[j].hi - S2[j].lo)$  then
    AppendSeq ( $S, S1[i].nodelist, S1[i].hi + base1, S1[i].lo + base1$ )
     $base2 = S1[i].lo$ 
     $i++$ 
  else
    AppendSeq ( $S, S2[j].nodelist, S2[j].hi + base2, S2[j].lo + base2$ )
     $base1 = S1[j].lo$ 
     $j++$ 
  end while
return  $S$ 
```

AppendSeq ($S, nodelist, hi, lo$):

```

 $E = (nodelist, hi, lo)$  // new element to append to  $S$ 
 $i = |S|$ 
while  $i \geq 1$  and  $(E.hi \geq S[i].hi \text{ or } E.lo \leq S[i].lo)$ 
   $E = (S[i].nodelist + E.nodelist, \max(S[i].hi, E.hi), E.lo)$  //  $S[i]$  is combined into  $E$ 
  remove  $S[i]$  from  $S$ 
   $i--$ 
end while
 $S = S + E$  //  $|S|$  is now  $i + 1$ 
```

Fig. 3. Procedure for finding an memory-optimal traversal of an expression tree

Node v	Optimal traversal $v.seq$
A	$\langle (A, 20, 20) \rangle$
B	$\langle (AB, 23, 3) \rangle$
C	$\langle (C, 30, 30) \rangle$
D	$\langle (CD, 39, 9) \rangle$
E	$\langle (CD, 39, 9), (E, 25, 16) \rangle$
F	$\langle (CD, 39, 9), (ABEF, 34, 15) \rangle$
G	$\langle (G, 25, 25) \rangle$
H	$\langle (GH, 30, 5) \rangle$
I	$\langle (CDGHABEFI, 39, 16) \rangle$

(a) The expression tree in Fig. 1

(b) Optimal traversals for subtrees

Fig. 4. Optimal traversals for the subtrees in the expression tree in Fig. 1

The amount of adjustment for an element from $S1$ ($S2$) equals the lo value of the last merged element from $S2$ ($S1$), which is kept in $base1$ ($base2$).

The **AppendSeq** procedure appends a new element specified by $nodelist$, hi , and lo to the given traversal S . Before the new element E is appended to S , it is combined with elements at the end of S whose hi is not higher than $E.hi$ or whose lo is not lower than $E.lo$. The combined element has the concatenated $nodelist$ and the highest hi but the original $E.lo$. Elements are combined to form larger indivisible units.

To illustrate how the algorithm works, consider the expression tree shown in Fig. 1 and reproduced in Fig. 4(a). We visit the nodes in a bottom-up order. Since A has no children, $A.seq = \langle (A, 20, 20) \rangle$ (for clarity, we write $nodelists$ in a sequence as strings). To form $B.seq$, we take $A.seq$ and append a new element $(B, 3 + 20, 3)$ to it. The **AppendSeq** procedure combines the two elements into one, leaving $B.seq = \langle (AB, 23, 3) \rangle$. Here, A and B form an indivisible unit, implying that B must follow A in some optimal traversal of the entire expression tree. Similarly, we get $E.seq = \langle (CD, 39, 9), (E, 25, 16) \rangle$. For node F , which has two children B and E , we merge $B.seq$ and $E.seq$ by the order of decreasing $hi-lo$ difference. So, the elements merged are first $(CD, 39, 9)$, then $(AB, 23 + 9, 3 + 9)$, and finally $(E, 25 + 3, 16 + 3)$ with the adjustments shown. They are the three elements in $F.seq$ after the merge as no elements are combined so far. Then, we append to $F.seq$ a new element $(F, 15 + 19, 15)$ for the root of the subtree. The new element is combined with the last two elements in $F.seq$. Hence, the final content of $F.seq$ is $\langle (CD, 39, 9), (ABEF, 34, 15) \rangle$, which consists of two indivisible units. The optimal traversals for the other nodes are computed in the same way and are shown in Fig. 4(b). At the end, the algorithm returns the optimal traversal $\langle C, D, G, H, A, B, E, F, I \rangle$ for the entire expression tree (see Fig. 2(c)).

The time complexity of this algorithm is $O(n^2)$ for an n -node expression tree because the processing for each node v takes $O(m)$ time, where m is the number of nodes in the subtree rooted at v . Another feature of this algorithm is that the traversal it finds for a subtree T' is not only optimal for T' but must also appear as a subsequence in some optimal traversal for any larger tree that contains T' as a subtree. For example, $E.seq$ is a subsequence in $F.seq$, which is in turn a subsequence in $I.seq$ (see Fig. 4(b)).

4 Conclusion

In this paper, we have considered the memory usage optimization problem in the evaluation of expression trees involving large objects of different sizes. This problem can be found in many practical applications such as scientific calculations, database query, and data mining, for which the data objects can be so large that it is impossible to keep all of them in memory at the same time. Hence, it is necessary to allocate and deallocate space for the data objects dynamically and to find an evaluation order that uses the least memory. We have proposed an efficient algorithm that finds an optimal evaluation in $O(n^2)$ time for an expression tree containing n nodes. Also, we have described some interesting properties of the problem and the algorithm.

References

1. A. W. Appel and K. J. Supowit, *Generalizations of the Sethi-Ullman algorithm for register allocation*, Software—Practice and Experience, 17 (6), pp. 417–421, June 1987.
2. W. Aulbur, *Parallel implementation of quasiparticle calculations of semiconductors and insulators*, Ph.D. Dissertation, Ohio State University, Columbus, October 1996.
3. M. S. Hybertsen and S. G. Louie, *Electronic correlation in semiconductors and insulators: band gaps and quasiparticle energies*, Phys. Rev. B, 34 (1986), pp. 5390.
4. C. Lam, P. Sadayappan, and R. Wenger, *On optimizing a class of multi-dimensional loops with reductions for parallel execution*, Parallel Processing Letters, Vol. 7 No. 2, pp. 157–168, 1997.
5. C. Lam, P. Sadayappan, and R. Wenger, *Optimization of a class of multi-dimensional integrals on parallel machines*, Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
6. C. Lam, P. Sadayappan, D. Cociorva, M. Alouani, and J. Wilkins, *Performance optimization of a class of loops involving sums of products of sparse arrays*, Ninth SIAM Conference on Parallel Processing for Scientific Computing, March 1999.
7. C. Lam, *Performance optimization of a class of loops implementing multi-dimensional integrals*, Technical report no. OSU-CISRC-8/99-TR22, Dept. of Computer and Information Science, The Ohio State University, Columbus, August 1999.
8. I. Nakata, *On compiling algorithms for arithmetic expressions*, Comm. ACM, 10 (1967), pp. 492–494.
9. H. N. Rojas, R. W. Godby, and R. J. Needs, *Space-time method for Ab-initio calculations of self-energies and dielectric response functions of solids*, Phys. Rev. Lett., 74 (1995), pp. 1827.
10. R. Sethi, J. D. Ullman, *The generation of optimal code for arithmetic expressions*, J. ACM, 17(1), October 1970, pp. 715–728.
11. R. Sethi, *Complete register allocation problems*, SIAM J. Computing, 4(3), September 1975, pp. 226–248.

Resource Usage Modelling for Software Pipelining

V. Janaki Ramanan¹ and R. Govindarajan^{1,2}

¹ Supercomputer Education and Research Centre

² Department of Computer Science and Automation

Indian Institute of Science

Bangalore 560 012, India

{ramanan@rishi.serc, govind@serc}.iisc.ernet.in

Abstract. In this paper we propose two optimization techniques for resource modelling in software pipelining. The first technique is the extension of our *grouping* technique to reduce the space overhead in automaton based software pipelining methods. This technique exploits the symmetry in the states of the automaton to produce the Group Automaton (GA) with reduced number of states. Our second technique is the Conjugate Offset method which eliminates the “symmetric” offset sets from the set of offset sets produced by the Reduce MS-State Diagram approach. Our experimental results reveal that the proposed optimizations result in significant reduction in the space requirements.

1 Introduction

Instruction scheduling is an important phase in compiling for modern processors to exploit higher instruction level parallelism (ILP) [4]. The schedule constructed by an instruction scheduler must obey both data dependency constraints imposed by the program semantics and resource constraints imposed by the target architecture. The resource constraints of the target architecture include the resource usage patterns of various instruction classes. A resource usage model helps to answer the question: “Given a target machine and a partial schedule, can a new instruction be placed in the partial schedule in the time slot t without causing any resource conflict?” Thus in order to speed up the instruction scheduling method the resource usage models also need to be very efficient.

In this paper, we examine the resource models in the context of Software Pipelining [7,12,13]. Software pipelining is an aggressive instruction scheduling technique to exploit parallelism in loops. Many of the resource modelling techniques developed for basic instruction scheduling such as reservation table based methods [2,3] have been extended to software pipelining. Modulo Scheduled State Diagram (MS-SD) [5] is one such approach which relies on automaton based techniques. Two distinct approaches can be followed in using the state

diagrams for modulo scheduling. In one approach, the state diagram is represented in the form of a transition table, and scheduling an instruction and updating the machine state involves a state transition. Whereas, in the second approach, legal initiation sequences are modelled as *offset sets* [6]. An offset set specifies the set of time steps in the repetitive kernel in which instructions can be scheduled without causing any resource conflict.

This paper deals with the above two approaches followed in state diagram based resource modelling. The space-time trade-off of state-diagram based and reservation table based approaches for basic instruction scheduling is discussed in [11]. However, in the context of software pipelining, we focus only on space reductions and leave the compile time considerations for future work. The first part of the paper deals with the FSA based resource models for software pipelining. Here we extend the *grouping* technique proposed in [11] for constructing the group automaton for modulo scheduling. This technique results in considerable reduction in the space required to store the automaton. The second part of the paper deals with the Offset set based approaches for software pipelining. These approaches use offset sets to reason about the schedulability of instructions at various time steps in the partial schedule. In this context we propose the Conjugate Offset method which eliminates the “symmetric” offset sets from the set of offset sets and thus reduces the space requirements to store the offset sets.

The paper is organized as follows. Section 2 describes finite state automaton based technique to construct the automaton. Section 3 extends the grouping technique for constructing the automaton for software pipelining. Experimental results comparing the performance of the various automaton based approaches are reported in Section 4. In Section 5 we discuss the Offset set based software pipelining technique. Finally, concluding remarks are provided in Section 6.

2 FSA Based Technique for Software Pipelining

FSA approach was used for resource modelling in the context of basic instruction scheduling in [8,9,1]. In this section, we extend this FSA approach for software pipelining. Before proceeding with this technique, we provide the necessary background.

Definition 1. A Collision Matrix C_I for the instruction class I is a bit matrix of size $n \times \ell'$, where n is the number of instruction classes and ℓ' is the longest repeat latency of an instruction class. The entry $C_I[J,t] = 1$ implies that if an instruction of class I is scheduled in the current cycle then an instruction of class J cannot be scheduled t cycles from the current cycle.

The notation $CM_I[J]$ is used to denote the Jth row in the collision matrix of instruction class I . This bit-vector, determines the *cross-collision* behaviour of J with respect to I . The construction of the FSA proceeds as in [9,1]:

1. The initial state is associated with a $n \times \ell'$ state matrix whose entries are all zeros.
2. For each instruction class I such that the $[I, 0]$ th entry in a state matrix is 0, the initiation of an instruction belonging to class i is legal and causes a state transition to a new state whose state matrix is obtained by bitwise OR-ing the current state matrix with the CM_I .
3. A cycle advancing (CA) transition can be made from any state, and the state matrix of the new state is obtained by left-shifting the current state matrix by one position.

The automaton is laid down in the form of a transition table which is used by the scheduler at the time of scheduling the instructions. The transition table is a two dimensional matrix where the rows correspond to the distinct states of the automaton while the columns correspond to the instructions classes.

A state diagram based approach for resource modelling has been proposed for modulo scheduling (MS-SD) [5,6] using the cyclic collision matrix (CCM) instead of CM. A CCM is similar to a CM except that these represent the self and the cross-collision behaviour of instruction in a schedule window of length II . In the construction of the MS-SD developed in [5,6] initiations corresponding to all latencies from 0 to $(II - 1)$ are examined from a given state. The new state matrix corresponding to an initiation of class I instruction after k cycles is obtained by left-rotating the current state matrix by k positions and bitwise OR-ing CM_I with it. The MS-SD construction does not require any explicit cycle advancing instructions.

Consider a machine model comprising of 4 instruction classes *viz.*, the Add, Shift, Mult and the Divide class. The *CCM* for these instruction classes for $II = 4$ are given in Figure 1. We use this as a motivating example to explain the various automaton based techniques. The partial MS-SD for the motivating example is shown in Figure 2(a). Contrast this with the traditional FSA approach discussed above, for simple basic block scheduling, where only column 0 is examined. Intuitively, it seems that extending the FSA approach to software pipelining would result in reduced space requirement to store the automaton, since only column 0 is examined and therefore the number of transitions would be less when compared to the MS-SD.

Next, we extend the traditional FSA approach to software pipelining. We call this alternative approach as MS-FSA (Modulo Scheduled-Finite State Automaton). The construction of MS-FSA is very similar to the traditional FSA except for the following fact:

- MS-FSA makes use of cyclic collision matrices, CCM , for the construction of the automaton while FSA approach uses the simple collision matrices.
- The pseudo operation $ms\text{-}CA$ is used in MS-FSA to advance the machine state by one cycle. This pseudo instruction $ms\text{-}CA$, causes the current state matrix to be left-rotated (rather than left-shifted) once.

The partial MS-FSA for the motivating example is shown in Figure 2(b). Due to lack of space, the reader is referred to [10] for details of an efficient implementation of the transition table for the MS-SD approach.

3 Group Automaton for Software Pipelining

In the case of FSA, the space required to store the transition table was found to be a few kilobytes of memory [1,11]. However, in the case of MS-FSA, the size

$$\begin{array}{llll}
 \begin{matrix} a \\ s \\ m \\ d \end{matrix} \left[\begin{matrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{matrix} \right] &
 \begin{matrix} a \\ s \\ m \\ d \end{matrix} \left[\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{matrix} \right] &
 \begin{matrix} a \\ s \\ m \\ d \end{matrix} \left[\begin{matrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \right] &
 \begin{matrix} a \\ s \\ m \\ d \end{matrix} \left[\begin{matrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \right] \\
 CM_{Add} & CM_{Shift} & CM_{Mult} & CM_{Div}
 \end{array}$$

Fig. 1. Collision matrices for Add, Shift, Mult and Div classes for $II = 4$

of automaton is much larger and it increases drastically with the increase in II . In this section, we extend the *grouping technique* developed in [11] to software pipelining to reduce the size of the automaton while retaining its benefits. Before we proceed, we briefly explain the idea behind grouping technique. It is observed from the MS-FSA shown in Figure 2(b) that the states $F1$ and $F2$ are *symmetric* in the sense that their state matrices are similar except for fact that the row 1 and row 2 in $F1$ are interchanged in the state matrix of $F2$. Further, the successor states of the symmetric states are also symmetric (in this case successor states are the same). In the construction of group automaton, these symmetric states are not generated. This symmetry of states in the automaton can be exploited by grouping of instruction classes Add and Shift. It has been formally established that if the criteria for grouping are satisfied by the collision matrices, then the corresponding criteria for the CCMs are also satisfied. Refer to [10] for details on these criteria.

Construction of group automaton for software pipelining, henceforth referred to as, MS-GA (Modulo Scheduled Group Automaton), is very much similar to the GA. The differences between GA and MS-GA constructions are same as that between FSA and MS-FSA, namely (i) the use of CCMs instead of CMs and

(ii) the left rotation in ms-CA transition. The partial group automaton for the motivating example is shown in Figure 2(c).

Finally, we integrated the grouping technique with the MS-SD approach to obtain the modulo-scheduled group state-diagram (MS-GSD). In this technique we used the grouping criteria to group the instruction classes and used state-diagram based construction procedure [5] to construct the state diagram for the grouped classes.

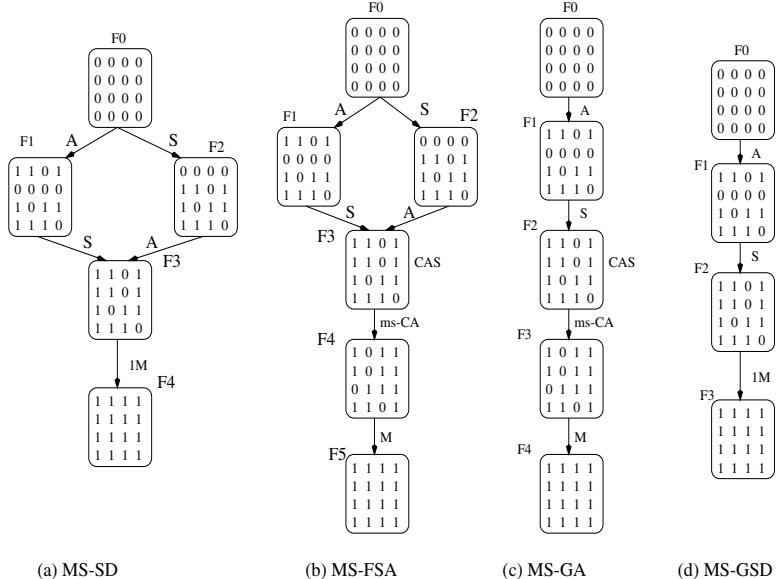


Fig. 2. Different Automatons/State Diagrams for the Motivating Example

4 Performance Comparison

In this section, we report the performance of the various automaton and state diagram based approaches in terms of the space required to store the transition tables for the motivating example discussed in Section 2. From Table 1, we observe that MS-GA achieves significant reduction (factor of 2 to 3 for II values between 6 to 10) in terms of both number of states and the space requirements when compared to MS-FSA. This is due to the fact that large number of states are produced in MS-FSA out of which many are suppressed in MS-GA¹. There is a significant reduction in the number of states in the MS-SD compared to

¹ It is to be noted that for $II > 12$, the number of states in MS-FSA were quite large. Hence, for the sake of comparison, we restricted the range of II to 12.

MS-FSA. The reduction in the number of states is more than 50% in most of the cases. However, for higher values of II , the space requirements of MS-SD is more (around 22%) than that of MS-FSA. This is because of the fact that there are more number of legal transitions from each state in MS-SD than in MS-FSA. Compared to MS-SD and MS-FSA, the number of states and the space requirements in MS-GSD is significantly low. The reduction in the number of states is roughly by a factor of 7 for value of II in the range 10 to 12. The reduction achieved in space is by a factor of 3. Though the space requirement for MS-GSD is increasing with II , the requirement is not as high as in MS-FSA and MS-SD. Thus, we conclude that the MS-GSD performs better than both MS-FSA and MS-SD.

Table 1. Performance Comparison of Different Approaches

Period (II)	MS-SD		MS-FSA		MS-GA		MS-GSD	
	No. of States	Space (in KB)						
4	22	0.38	39	0.76	30	0.586	14	0.23
5	72	1.54	146	2.85	86	1.679	38	0.78
6	168	4.59	348	6.79	200	3.906	83	2.16
7	541	16.60	1170	22.85	617	12.051	235	6.98
8	1463	552.39	3175	62.01	1558	30.429	605	20.74
9	4180	169.23	9109	177.91	3904	76.251	1539	60.35
10	11331	514.38	24638	481.21	9637	188.223	3809	167.48
11	30987	15550.38	67057	1309.71	24069	470.098	9484	460.50
12	84992	4649.93	184104	3595.78	60300	1177.738	23739	1258.69

5 Offset Set Based Software Pipelining Techniques

In this section, we discuss the Offset set based techniques which can be used in high performance schedulers which perform some form of backtracking [12] to reverse the scheduling decisions made at an earlier point of time. The optimization proposed here is for pipelines where instruction classes do not share resources. An extension of this for such cases is left for future work. Hence we consider a pipeline which supports a single function or an instruction class. A path $S0 \xrightarrow{p_1} S1 \xrightarrow{p_2} S2 \dots \xrightarrow{p_k} Sk$ in the MS-State diagram corresponds to a permissible or legal latency sequence $\{p_1, p_2, \dots, p_k\}$. The latency sequence represents $k + 1$ initiations that are made at time steps 0, p_1 , $(p_1 + p_2)$, \dots , $(p_1 + p_2 + \dots + p_k)$. In modulo scheduling, these time steps correspond to the *offset values* 0, p_1 , $(p_1 \oplus p_2)$, \dots , $(p_1 \oplus p_2 \oplus \dots \oplus p_k)$ in a repetitive kernel, where \oplus refers to addition modulo II [6]. A set comprising of these offset values is referred to as

an *Offset* set. Lastly, in [6], a software pipelining method which uses the set of offset sets as a resource usage model has been proposed. Further, as only distinct offset sets need to be considered, redundant paths which result in duplicate offset sets can be eliminated in the reduced MS-State Diagram method [6].

5.1 Conjugate Offset Set Method

Consider the collision vector for an instruction class I as shown below:

$$CM_I = [1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0];$$

Let the initiation interval $II = 9$. The offset sets for this instruction class are: $\{0, 3, 6\}$, $\{0, 1\}$, and $\{0, 8\}$. Although the offset sets $\{0, 1\}$ and $\{0, 8\}$ are distinct, one can be used to derive the other. Thus only one of the two offset sets (known as *conjugate pairs*) need to be stored. This idea is formalized below.

Definition 2. *The conjugate Ω_c of an offset set $\Omega = \{o_0, o_1, o_2, \dots, o_k\}$ is defined as, $\Omega_c = \{(II \ominus o_0), (II \ominus o_1), (II \ominus o_2), \dots, (II \ominus o_k)\}$, where \ominus represent subtraction modulo II .*

Lemma 1. *If $\Omega = \{o_0, o_1, o_2, \dots, o_k\}$ is the offset set corresponding to a path P , then the conjugate of this offset set Ω_c is also an offset set and there exists a path corresponding to Ω_c in the Reduced MS-State Diagram.*

For proof of this lemma, refer to [10]. From this lemma we have established that for every offset set Ω in the Reduced MS-State diagram, there also exists a conjugate offset set Ω_c . Hence, it suffices to store only one of them and infer the other at the time of scheduling.

5.2 Performance of Conjugate Offset Set Method

In this section we compare the space requirements in terms of number of offset sets, for our Conjugate offset set method and the Reduced MS-State Diagram. We evaluated the two approaches for the function unit model shown in Table 2(a). Table 2(b) shows the results for the two approaches. It can be seen from Table 2(b), that the Conjugate Offset set Method eliminates almost 50% of the offset sets for the range of II values. This is because of the fact that, for every offset set there exists a conjugate offset set in the Reduced MS-State Diagram (refer to Lemma 1). It is observed that, certain offset sets are conjugates of themselves. Such offset sets are not eliminated in our approach. As a result of this, reduction in the number of offset sets is slightly less than 50%. It is to be noted that this efficiency in space is achieved at the expense of additional computational requirements at time of scheduling.

Period (II)	# Offset Sets			%age Reduction
	Red.	MS-SD	COM	
30	1224	616	49.7	
31	1601	812	49.3	
32	2208	1112	49.6	
33	2809	1420	49.4	
34	3689	1852	49.8	
35	4930	2486	49.6	
36	6396	3208	49.8	
37	8651	4353	49.7	
38	11495	5765	49.8	
39	15182	7627	49.8	
40	20367	10201	49.9	
41	26642	13369	49.8	

(a) Example FU

(b) Space Requirements

Table 2. Performance of Conjugate Offset Method

6 Conclusions

In this paper, we compare various automaton based resource modelling techniques for software pipelining. We extended the *grouping technique* to software pipelining and compared its performance, in terms of space required to store the state transition table, with the existing approaches. We observe that:

1. MS-GA performs better than MS-FSA in terms of space requirements for all the values of II .
2. The space requirements for MS-SD is lower than MS-FSA for small values of II and for a particular implementation of the transition table. But for higher values of II , MS-FSA performs better even though the number of states in MS-FSA is higher than in MS-SD.
3. The MS-GSD model, the combined technique, performs significantly better in terms of both space and the number of states when compared with MS-FSA, MS-SD or MS-FSA.
4. Finally, our Conjugate Offset set method for the case of single function unit reduces the space requirements for Reduced MS-SD by 50% by eliminating conjugate offset sets. However, the space efficiency achieved in this method is at the expense of additional computational requirements incurred at the time of scheduling.

Acknowledgements

We would like to thank V. Santhosh Kumar for suggesting an elegant implementation of the Conjugate Offset Set Method and for proof-reading this paper.

References

1. V. Bala and N. Rubin. Efficient instruction scheduling using finite state automata. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 46–56, Ann Arbor, MI, Nov. 1995.
2. J. C. Dehnert and R. A. Towle. Compiling for Cydra 5. *Jl. of Supercomputing*, 7:181–227, May 1993.
3. J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Trans. on Computers*, 7(30):478–490, July 1981.
4. P. B. Gibbons and S. S. Muchnick. Efficient instruction scheduling for a pipelined architecture. In *Proc. of the SIGPLAN '86 Symp. on Compiler Construction*, pages 11–16, Palo Alto, CA, June 25–27, 1986. .
5. R. Govindarajan, E. R. Altman, and G. R. Gao. Co-scheduling hardware and software pipelines. In *Proc. of the Second Intl. Symp. on High-Performance Computer Architecture*, pages 52–61, San Jose, CA, Feb. 3–7, 1996.
6. R. Govindarajan, N.S.S. Narasimha Rao, E.R. Altman, and G. R. Gao. Software pipelining using reduced ms-state diagram. In *Proc. of the Merged 12th Intl. Parallel Processing Symposium and 9th Intl. Symposium on Parallel and Distributed Processing*, Orlando, FL, March 1998.
7. M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the SIGPLAN '88 Conf. on Programming Language Design and Implementation*, pages 318–328, Atlanta, Georgia, June 22–24, 1988.
8. T. Muller. Employing finite automata for resource scheduling. In *Proc. of the 26th Intl. Symposium on Microarchitecture*, December 1993.
9. T. A. Proebsting and C. W. Fraser. Detecting pipeline structural hazards quickly. In *Conf. Record of the 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 280–286, Portland, OR, Jan. 17–21, 1994.
10. V. Janaki Ramanan. Efficient resource usage modelling, MSc(Engg) thesis, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India, 1999.
11. V. Janaki Ramanan and R Govindarajan. Resource usage models for instruction scheduling: Two new models and a classification. In *Proceedings of the 1999 ACM SIGARCH International Conference on Supercomputing (ICS-99)*, June 1999.
12. B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Ann. Intl. Symp. on Microarchitecture*, pages 63–74, San Jose, CA, Nov. 1994.
13. B. R. Rau and C. D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *Proc. of the 14th Ann. Microprogramming Workshop*, pages 183–198, Chatham, MA, Oct. 12–15, 1981.
14. C. Zhang, R. Govindarajan, S. Ryan, and G. R. Gao. Efficient state-diagram construction methods for software pipelining. In *Proc of Compiler Construction Conference*, March 1999.

An Interprocedural Framework for the Data and Loops Partitioning in the SIMD Machines

Jin Lin, Zhaoqing Zhang, Ruliang Qiao, and Ningning Zhu

Institute of Computing Technology
Chinese Academy of Sciences
Beijing 100080, China
Email: {ln,zzq,qrl}@act.ict.ac.cn

Abstract. This paper presents a compiler algorithm for the distributed memory machines that automatically finds an optimal partitioning for the arrays and loops in a whole program at the presence of procedural calls. In this framework, the whole program is formulated as interprocedural iterations and data graph (IDG), and the interprocedural search space propagation and synthesis technique is proposed to determine an optimal search space of the loops and data partitioning. Based on the performance estimation model, the interprocedural candidate selection algorithm using dynamic programming technique is proposed to determine an optimal partitioning of the loops and data in the program. These ideas are adopted into the design and implementation of a parallelizing compiler targeted on the SIMD machines. We demonstrate the efficiency of this framework with encouraging experimental results.

1 Introduction

Memory subsystem efficiency is critical to achieving high performance on parallel machines. A key issue in programming these machines is selecting the loops and data partitioning across the processors of the machines.

Our work focus on the data partitioning analysis for a SIMD machine with a large global memory (GM) and a set of small local memory (LM), where each corresponds to a processor element (PE). The objective of our framework is to determine the array and loops partitioning in a whole program having multiple procedures, so that the resultant performance gains contributed by these distributions is maximum.

Many researchers have addressed the problem of automatically selecting the optimal data layout. Li, Chen [1] use the Component Affinity Graph (CAG) to represent alignment preference, and use a heuristic algorithm to solve it. Kremer [2] use the 0-1 integer programming to solve the data layout problem, which is implemented in the D programming tools. However, these algorithms focus on intraprocedural analysis. Recently, there are some works on the interprocedural distribution analysis. Gupta and Krishnamurthy [3] propose the data distribution selection algorithm for global arrays using interprocedural analysis and dynamic programming techniques. However, they don't consider the array reshaping translation between the formal arrays and actual arrays.

Our work complements the work of Anderson [4]. The work of Anderson does a global analysis across procedures using affine transformations. However, it has the following difference. It first tries to find a communication free partitioning for each procedure in bottom up manner, and will lead to a less optimal solution in some case. Our method is to use search space propagation and synthesis technique to produce an optimal partitioning search space based on the detection of structure communication, and use dynamic programming method under a cost model to find a communication minimal partitioning for the entire program. The proposed strategy has been incorporated in a parallelizing C compiler to determine the distributed code generated in the program.

The outline of the rest of the paper is as follows. Section 2 introduces the basic concepts required to understand the framework. Section 3 provides an interprocedural framework for determining the optimal partitioning for the data and loops partitioning. Section 4 presents the results obtained by testing the proposal algorithm on several classic DSP applications, and followed by the conclusions.

2 Preliminaries

This section introduces some basic concepts about the memory layout and loop partitioning theory. The iteration space τ for a statement S of depth d in a non-perfect loop nest consists of the values $(i_1 \dots i_d)$ of surrounding loop variables. An array of dimension m defines an array space α , an m dimensional rectangle. Similarly, the one dimensional processor element array in the SIMD machine defines a processor space ρ .

In our discussion, the affine mapping of statement space τ onto ρ is $IMAP(\vec{i}) = T_\tau \vec{i} + \phi$, where \vec{i} is the iteration vector in the statement space τ , T_τ is an elementary basis vector and ϕ is a constant. Similarly, the mapping of array space α onto ρ is $DMAP(\vec{a}) = T_\alpha(\vec{a}) + \varphi$, where T_α is an elementary basis vector and φ is a constant.

3 Interprocedural Framework of Data and Loops Partitioning

In this section, we present an interprocedural framework, which automatically determine an optimal partitioning for the loops and arrays in a whole program with multiple procedural calls. It can also be employed as part of a unified technique for optimizing the communication and parallelism in distributed memory machines.

3.1 Graph Formulation

In our interprocedural framework, a program consisting a set of procedures is represented using a directed graph G_c , called call multi graph (CMG), where

$G_c = \langle V, E \rangle$. Each edge $e: V_p \rightarrow V_q$ in E represents a call site in procedure P where procedure Q is invoked. The edge e is associated with the binding relations among the array parameters and global arrays.

For each procedure in the program, we propose another graph data structure called iterations and data graph (IDG) to determine the resultant mapping functions for the loops and data arrays accessed in the procedure. The IDG for a procedure P is a weighted, undirected graph $G_p = \langle V_\tau, V_\alpha, E \rangle$, where a vertex in V_τ represents a loop nest in procedure P , and a vertex in V_α represents an array in procedure P . Each vertex in V_τ and V_α is associated with a set of mapping functions. The edges in E connect the arrays and the loop nests where the arrays are accessed.

3.2 Mapping Functions Search Space Construction

In this section, we propose an iterative search technique to derive an optimal set of mapping functions for the loops and arrays in the program based on the detection of structured communication.

The conditions for detecting structured communication are in the following, and they can be deduced from [5]. Given a statement S of depth d and an array with array expression $G\vec{i} + \vec{\delta}$, let $T_\tau\vec{i} + \phi$, $T_\alpha\vec{a} + \varphi$ be the mapping function of the iteration space and array space separately. The broadcast communication occurs if the following condition is satisfied:

$$GT_\tau^T = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \text{ where the matrix } \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \text{ is a } 1 \times d \text{ matrix,}$$

And the shift communication occurs if the following conditions are satisfied:

$$(1) T_\tau = T_\alpha G', \text{ where } G' = \begin{pmatrix} 0 & \dots & g_{1k} & \dots & 0 \\ \dots & & & & \\ 0 & \dots & g_{dk} & \dots & 0 \end{pmatrix}, T_\tau \text{ is the } k\text{th elementary basis vector.}$$

$$(2) T_\tau \neq T_\alpha G.$$

The following is the method in more detail. The mapping functions for the loops and arrays in the IDG are initiated according to the number of the array dimensions and the number of parallel loops in the loop nest. The parallelism we exploit can be *forall* or *pipelining* parallelism. Then an iterative improvement is followed.

The intraprocedural search space construction phase process the vertex in the IDG in the order of the weights. For each vertex, we first perform local search space synthesis to reserve the mapping function in its search space so that the communication between the vertex and its neighboring vertices are of structured communication or the communication cost is the minimum value. Then the local search space propagation follows, that is, for each neighboring vertex, we only reserve the mapping functions in its search space so that the communication between the vertices is of structured communication. This will continue until the search space of all the vertices in the IDG don't change.

After the mapping functions of the formal array parameters and global arrays accessed in a procedure are calculated in the intraprocedural phase, this information is made available to each of the call sites in the CMG.

The IDG of a procedure in a program can be extended to an interprocedural IDG $G_p = \langle V_\tau, V_\alpha, V_p, E \rangle$, where the vertices V_p represent the array parameters and global arrays. And the edges between the formals and the actuals are added into the set E to map the access matrix and mapping functions of the formals to actuals for array reshaping, and vice versa. The array reshaping we handle is one or more complete dimensions of an array are passed as a parameter. Similarly, the edges between the global arrays accessed in the caller and callee procedures are added into E .

The interprocedural search space construction phase processes each procedure vertex in the CMG in the order of the weights. For each vertex procedure in the CMG, we construct the corresponding interprocedural IDG, and perform global search space synthesis. The global synthesis is to update the mapping functions of all the vertices using the iterative method of intraprocedural phase according to the change of search space of array parameters and global array in the callee procedures and caller procedures. Then the global search space propagation follows. It propagates the change of the search space of the array parameters and the global arrays to the caller and callee procedures. That is, for each neighboring procedure in the CMG, the mapping functions search space for all the vertices are updated using the iterative method of intraprocedural phase. The algorithm terminates after the mapping functions of the vertices in each interprocedural IDG won't change.

3.3 Mapping Functions Candidate Selection

After the search space construction phase is finished, the number of the mapping function candidates in the search space may be more than one. In order to determine an optimal mapping function from the set of candidates, it is necessary to evaluate each candidate by providing a relative measure of the benefits ensuing from using the corresponding mapping functions. These merit values are used to select the final mapping functions across the procedures. For instance, given an interprocedural IDG for procedure *main*, the mapping functions selection problem is cast as selecting an optimal mapping function for each loop nest and array such that the objective function

$$\sum_{i=1}^n T_i$$

is minimized, where n is the number of loop nests in procedure *main*.

Our strategy to solve the interprocedural mapping functions selection problem is to solve the mapping selection problem for each procedure and propagate the effects to all of its call sites along valid execution path. The solution to the global problem is finally obtained by combining the solutions of each local problem.

The interprocedural candidate selection phase involves a bottom up traversal of the procedures in the CMG. For each procedure in the program, we first generate all the combinations of the mapping functions for the array formal parameters and global arrays. For each combination, we perform the iterative method on the IDG to select an optimal mapping function for each vertex, and calculate the corresponding performance merits. This information is made available to the call sites by the depth first traversal of the CMG. The summary information from the callee procedure is used at the related call sites to perform the intraprocedural candidate selection for the

caller procedure. The algorithm terminates after the mapping functions selection determines the mapping functions for the loops and arrays for the procedure *main* in the program. At the time of code generation, the mapping functions of the arrays and loop nests can be obtained by starting with the candidate mapping functions that has the minimum execution time at the start node of *main* procedure. After each procedure in the CMG is processed in top down order, we get the final mapping function for the arrays and loop nests for each procedure in the program.

4 Experimental Results and Conclusions

We have implemented a parallelizing C compiler VCC for the SIMD machine. The proposal algorithm has been integrated in the part of the compiler to understand the extent to which our algorithm impacts the performance of the DSP applications running on the simulator (16 PEs). We select the five DSP benchmarks: MM(Matrix Multiply), Fir Filter, EC(Echo Cancellation), VD(Viterbi Decoder) and RR(Rake Receiver) and give some experimental results to show the effectiveness of the algorithm using.

	MM	FIR	EC	Viterbi	Rake
Serial Version	334962	137443	1399862	2351313	1071962
Global Analysis	5125	3465	35004	42756	23257
Hand Code	4576	2407	24016	42188	18266

Table 1. Running time in cycles for different application using different compiling methods

In summary, the approach can determine an optimal arrays and loops partitioning in a whole program with multiple procedural calls. This framework has been implemented in a parallelizing C compiler targeted on the SIMD machine. The working of this algorithm has been demonstrated for some DSP applications.

References

1. Li, J., Chen, M.: Index Domain Alignment: Minimizing Cost of Cross-Referencing between Distributed Arrays. Proc. of the 3rd Symposium on the Frontiers of Massively Parallel Computation, College Park MD (1990) 424-433
2. Kremer, U.: Automatic Data Layout for Distributed Memory Machines. Ph.D. thesis, Rice University (1995)
3. Gupta, S.K.S., Krishnamurthy, S.: An Interprocedural Framework for Determining Efficient Data Redistributions in Distributed Memory Machines. Proc. of the 6th Symposium on the Frontiers of Massively Parallel Computing, Annapolis Maryland (1996) 233-240
4. Anderson, J.: Automatic Computation and Data Decomposition for Multiprocessors. Ph.D. thesis, Stanford University (1997)
5. Dion, M., Randriamro, C., Robert, Y.: Compiling Affine Nested Loop: How to Optimize the Residual Communications after the Alignment Phase. J. of Parallel and Distributed Computing, 38 (1996) 176-187

Tiling and Processors Allocation for Three Dimensional Iteration Space

H.Bourzoufi, B.Sidi-Boulenouar, R.Andonov

Université de Valenciennes, LAMIIH-ROI, UMR 8530,
`{bourzouf,sidi,andonov}@univ-valenciennes.fr`
ISTV - Le mont Houy - BP 311 - 59304 Valenciennes Cedex

Abstract. We study in this paper the optimal tiling and processors allocation for loops of depth three defined in parallelepiped shaped iteration space. The particularity of the considered class is that the associated dependencies allow orthogonal tiling. We compare ring vs. grid architecture and provide exhaustive numerical experiments on distributed memory machine (Intel Paragon). We also apply the obtained results to solve huge instances of the Bidimensional Knapsack Problem (BKP), which is characterized by 2D non-uniform recurrences.

1 Introduction

Tiling the iteration space is a common method for improving the performance of parallel loops programs executed on distributed memory machines (*DMM*) (see [1, 2]). A tile is a collection of iterations to be executed as a single unit. Processors communication and synchronization occur only once per tile. Optimal tiling consists in determining the optimal parameters of the tile (shape and size) which minimize the execution time. The problem has been largely studied [2, 3, 5] in the case of 2D-iteration space and also, but weaker, in the case of 3D-iteration space. The main purpose of this paper is to reconsider the existing in the literature models [4, 5], to provide convincing experimental answers for some arguable theoretical points and above all, to give the programmer some useful hints in order to provide an efficient parallel code. The problem that we address can be formulated as follows: computing for all $\mathbf{j} = [j_1, j_2, j_3]^T \in Dom \subset Z_+^3$ a function, $Y[\mathbf{j}]$, given by: $Y[\mathbf{j}] = f(Y[\mathbf{j} - \mathbf{d}_1], Y[\mathbf{j} - \mathbf{d}_2], \dots, Y[\mathbf{j} - \mathbf{d}_h])$ with appropriate boundary conditions in a domain $Dom = \{\mathbf{j} : 0 < j_1 \leq m_1, 0 < j_2 \leq m_2, 0 < j_3 \leq m_3\}$, also called iteration space. f is a function computed at point \mathbf{j} in a single unit time, and $Y[\mathbf{j}]$ is its value at this point. The iteration space is a parallelepiped and the dependencies vectors d_i have non-negative components. In this case orthogonal tiling is possible, (i.e. tiles whose boundaries are parallel to the domain boundary are valid). The remainder of this paper is organized as follows: at first we summarize the main outlines of our model to determining the optimal tile size in 3D space. Then we review this model in order to study different processors mappings. Finally, we extend our study in a particular case of dynamic dependencies and apply our technique for solving the BKP problem.

2 Tiling 3-dimensional iteration space

This section describes tiling techniques for 3D iteration space. More details about the theoretical basis of this section can be found in [7]. The iteration space is partitioned into orthogonal tiles (rectangular parallelepipeds) of size $x_1 \times x_2 \times x_3$, where x_1, x_2, x_3 are considered as free variables. The tile graph of $\left\lceil \frac{m_1}{x_1} \right\rceil \times \left\lceil \frac{m_2}{x_2} \right\rceil \times \left\lceil \frac{m_3}{x_3} \right\rceil$ nodes is mapped over a grid of $P = p_1 \times p_2$ processors by performing multiple passes if necessary. The total running time is expressed as a function of the tile size whose optimal value is obtained by minimizing this function.

2.1 Execution time formulae

Our model introduces the notions of *period* \mathcal{P}_t and the *latency* \mathcal{L} , which are defined respectively as the time elapsed between corresponding instructions of two successive tiles mapped to the same processor (resp. adjacent processors) (see [5, 7]). The tile graph dependencies are given by the identity matrix I .

Mapping tile graph onto a grid of processors : We distribute the tiles on $p_1 \times p_2$ processors arranged in a grid. We choose the axis j_3 (parallel to edge m_3) as direction of projection; a tile (i, j, k) is distributed on $P_{imodp_1, jmodp_2}$ (because multiple passes are allowed). We use \mathcal{L}_i to denote the latency in the i -th axis. Along the axis of the projection, we have $\mathcal{L}_3 = \mathcal{P}_t$. The total execution time is given by the following formula (see [7]):

$$T(\mathbf{x}) = \frac{\mathcal{P}_t(\mathbf{x})}{P} \prod_{i=1}^3 \frac{m_i}{x_i} + (p_1 - 1)\mathcal{L}_1(\mathbf{x}) + (p_2 - 1)\mathcal{L}_2(\mathbf{x}) \quad (1)$$

where $\mathbf{x} \in X = \{\mathbf{x} \in R^3 : 1 \leq x_i \leq \frac{m_i}{p_i}, i = 1, 2 ; \quad 1 \leq x_3 \leq m_3\}$ and x_i are integer.

Mapping tile graph onto a ring of processors : Because we consider the class of orthogonal dependencies it is possible to map the tile graph on a ring of processors, by performing two projections along two different dimensions. Consider the axes which are parallel to the edges m_2 and m_3 ; the tile (i, j, k) is mapped on processor P_{imodp} . The parameters which give the optimal tile size are the values of x_1 and $x_2 x_3$. Hence, a tile can be viewed as a rectangle whose size is $x_1 \times (x_2 x_3)$ (we set $x_3 = 1$ and leave x_2 as free variable). To find the optimal tile size, we simply apply the results from the 2D-case [5, 6]. The execution time in this case is given by :

$$T(x_1, x_2) = \frac{m_1}{x_1} \frac{m_2 m_3}{x_2} \frac{1}{P} \mathcal{P}_t + (P - 1)\mathcal{L}_1 \quad (2)$$

2.2 Execution time on DMM

For the grid version, each processor $P_{i,j}$ executes the loop : receive two messages from the processors $P_{i-1,j}$ and $P_{i,j-1}$, compute a tile and send result to processors $P_{i,j+1}$ and $P_{i+1,j}$. On the ring the code is similar, except the communications which are simply pipelined. In the DMM communication model [8],

the time to transmit message of l words between two processors is $\beta + l\tau_t$, where the β is the startup latency and τ_t is the transmission rate. Let τ_a the time to compute function f at any point. The *period* and *latency* can be instanciated as follows :

$$\mathcal{P}_t^{grid}(x) = 4\beta + x_1 x_2 x_3 \tau_a \quad \mathcal{P}_t^{ring}(x) = 2\beta + x_1 x_2 \tau_a$$

$$\mathcal{L}_i^{grid}(x) = \mathcal{P}_t^{grid}(x) + \frac{x_1 x_2 x_3}{x_i} \tau_t - 2\beta \quad \mathcal{L}_1^{ring}(x) = \mathcal{P}_t^{ring}(x) + x_2 \tau_t - \beta$$

2β and β are subtracted in the latency functions because we consider that the sender and the receiver are properly synchronized, the calls are overlapped. Substituting and simplifying in (1) and (2), we obtain :

$$T(\mathbf{x})^{grid} = \frac{m_1 m_2 m_3}{x_1 x_2 x_3} \left(\frac{4\beta}{P} + \frac{\tau_a x_1 x_2 x_3}{P} \right) + \sum_{k=1}^2 (p_k - 1) \left(2\beta + \tau_t \frac{x_1 x_2 x_3}{x_k} + \tau_a x_1 x_2 x_3 \right) \quad (3)$$

$$T(x)^{ring} = \frac{m_1 m_2 m_3}{x_1 x_2 P} (2\beta + x_1 x_2 \tau_a) + (P-1)(\beta + x_1 x_2 \tau_a + x_2 \tau_t) \quad (4)$$

The tile size problem consists in minimizing the functions (3) and (4) in the associated domain. For the grid we have [7].

$$(x_1^*, x_2^*, x_3^*)^{grid} = \begin{cases} \left(\frac{m_1}{p_1}, \frac{m_2}{p_2}, x_3^* \right) & \text{if } \lambda_1 > 0 \text{ (non cyclic solution)} \\ (x_1^*, x_2^*, 1) & \text{otherwise (cyclic solution)} \end{cases} \quad (5)$$

where $\lambda_1 = 4\beta m_3 p_1 p_2 - [\tau_a m_1 m_2 (p_1 + p_2 - 2) + \tau_t (m_2 p_1 (p_1 - 1) + m_1 p_2 (p_2 - 1))]$. Let v be the tile volume. The optimal values x_3^* , x_1^* , x_2^* and v^* are given by :

$$x_3^* = \sqrt{\frac{4\beta m_3 p_1 p_2}{\tau_a m_1 m_2 (p_1 + p_2 - 2) + \tau_t (m_2 p_1 \tilde{p}_1 + m_1 p_2 \tilde{p}_2)}} \quad (6)$$

$$x_1^* = \sqrt{\frac{p_1 - 1}{p_2 - 1} v^*}, \quad x_2^* = \sqrt{\frac{p_2 - 1}{p_1 - 1} v^*} \quad v^* = \sqrt{\frac{4\beta m_1 m_2 m_3}{\tau_a p_1 p_2 (p_1 + p_2 - 2)}} \quad (7)$$

Let us set $\lambda_2 = 2Pm_2m_3\beta - (P-1)m_1\tau_a$. For the ring we have [5, 6]:

$$(x_2^*, x_1^*)^{ring} = \begin{cases} \left[\sqrt{\frac{2Pm_2m_3\beta}{(P-1)m_1\tau_a}}, \frac{m_1}{P} \right] & \text{if } \lambda_2 > 0 \text{ (non cyclic solution)} \\ \left[1, \sqrt{\frac{2m_1m_2m_3\beta}{(P-1)P\tau_a}} \right] & \text{otherwise (cyclic solution)} \end{cases} \quad (8)$$

3 Cyclic vs. non cyclic solution

The above results show that when $\lambda_i (i = 1, 2)$ are not positive the optimal behavior is given by the block cyclic distribution. For the ring, λ_2 can be negative in the particular case when m_1 is very large comparing to the value $m_2 \times m_3$. The results obtained in case of a grid are summarized in table (1). To confirm the theoretical results, we have carried out exhaustive computational experiments on Intel Paragon for which we have $\tau_a = 1\mu\text{sec}$; $\beta = 37\mu\text{sec}$; $\tau_t = 0.01\mu\text{sec}$. We have chosen different instances of m_1, m_2 and m_3 in order to obtain various forms of iteration space and to generate in this way the both cases ($\lambda_1 \geq 0$ and $\lambda_1 < 0$). The column 4 shows the theoretical optimal execution time obtained by the non-cyclic solution. We compute x_3^* according to the formula (6). If $(x_3^* \geq 1)$, this solution is kept; otherwise x_3^* is fixed to one (col 5). In fact, this column gives the best time that can be obtain by one pass. The column 8 shows the theoretical optimal execution time obtained by the cyclic solution (to be considered only in the case $\lambda_1 < 0$). The column 9 gives the required number of passes. We note

that the predicated and the measured total execution time are very close to each other. We observe that for the instances I1 and I2 the best time is given by the non-cyclic solution. In all other cases, the cyclic solution is more efficient (note how significant is the gain in the case of I3 and I4). By analyzing the sign of λ_1 , we note that even in case of a cube ($m_1 = m_2 = m_3$), the optimal behavior is given by the block cyclic distribution. Thus, it is worth having the code for the block cyclic distribution but let us also note that this code is more complicated: the data communicated by the last processor in each pass have to be buffered.

1	2	3	4	5	6	7	8	9	10
Inst	λ_1	Non-Cyclic				Cyclic			
		Theoretical	Exp	Theoretical	Exp				
I1	1437.95	14.15	640.044	/	647.123	516	1073.53	/	/
I2	1150.98	17.96	256.60	/	267.68	410	259.534	/	/
I3	-10199.7	0.05	640.03	1083.76	/	516.02	656.01	24	664.23
I4	-244.23	0.02	37.15	41.31	/	241.53	31.16	31	32.17
I5	-609.13	0.03	384.64	409.313	/	554.15	384.46	49	389.75

Table 1. [$p_1 = 5, p_2 = 4$] [I1: $m_1 = m_3 = 5 \times 10^4, m_2 = 10$] [I2: $m_1 = m_2 = 5 \times 10^4, m_3 = 4 \times 10^4$] [I3: $m_1 = m_2 = 5 \times 10^4, m_3 = 10$] [I4: $m_1 = 3 \times 10^4, m_2 = 2 \times 10^3, m_3$] [I5: $m_1 = 5 \times 10^4, m_2 = 3 \times 10^3, m_3 = 100$]

4 Grid vs. Ring

In the previous section, we have shown that the same tile graph can be mapped on grid or ring of processors. The interesting question is what is the best architecture? This is a hard problem in the general case and some theoretical results have been recently given in [9]. Experimentally, we did not observe significant difference between the two architectures on the Intel Paragon (only 50 processors were available). However, the ring version is more interesting for some non-uniform dependencies. We applied for example dynamic programming approach for solving huge instances of the bidimensional knapsack problem [10] (BKP). This reduces to compute the following recurrence

$$f(i, j, k) = \{ \max(f(i-1, j, k), p_i + f(i-1, j - w_i, k - u_i)) \quad (9)$$

in the iteration domain $\mathcal{D} = \{(i, j, k) : 0 \leq i \leq m ; 0 \leq j \leq c_1 ; 0 \leq k \leq c_2\}$. The peculiarity of these dependencies is that the values w_i and u_i vary in large interval with any index i and any instance of the problem. For these reasons, it is not possible to guaranty local memory access and constant communication volume on a grid of processors. This can be done only on a ring when the mapping follows the axes of the non uniform dependencies (j and k). Applying the tiling technique we solved huge instances of BKP. For the lack of space we provide bellow the experimental result for only one instance Fig. 3 (a). It comes from the literature [10] and have real life application. Figure 3 (b) illustrates the obtained linear speedup for another instance.

5 Conclusion

This paper describes some applications of the tiling techniques in the case of 3D iteration space. We consider two aspects of this problem. First we compare the

cyclic vs. non-cyclic block distribution of the iteration space. We confirm recent theoretical results and experimentally illustrate the importance of the cyclic solution, which is often ignored in related works. We also study the performance of grid vs. ring architectures. We observe that the both architectures provide the similar performance on Intel Paragon. As application of the ring architectures we solve bidimensional knapsack problem. To the best of our knowledge, the parallelisation of this problem has been not addressed before in the literature. The obtained experimental results confirm the effectiveness of our approach.

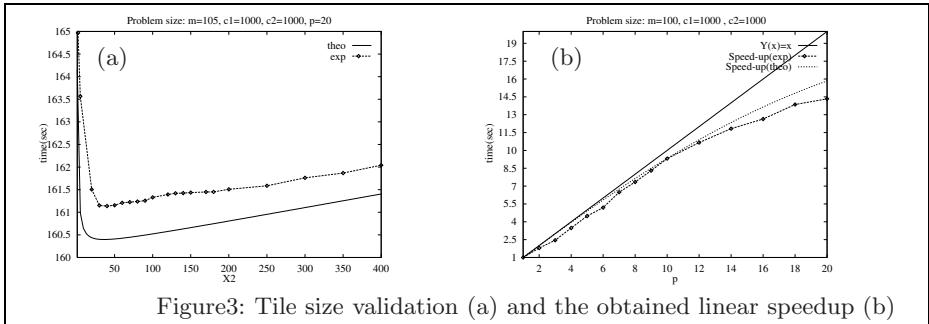


Figure3: Tile size validation (a) and the obtained linear speedup (b)

References

1. Hiranandani, S., Kennedy, K., Tseng, C.: Evaluating Compiler Optimizations for Fortran D. Journal of Parallel and Distributed Computing.(1994) 21:27–45
2. Irigoin, F., Triolet, R.: Supernode Partitioning. 15th ACM symposium on Principles of Programming Languages.(1988) 319–328
3. Ramanujam, J., Sadayappan, P.: Tiling Multidimensional Iteration Spaces for Non Shared-Memory Machines. Supercomputing 91.(1991) 111–120
4. Ohta, H., Saito, Y., Kainaga, M. Ona, H.: Optimal tile size adjustment in compiling general DOACROSS loop nests. International Conference on Supercomputing.(1995) 270–279
5. Andonov, R., Rajopadhye, S.: Optimal orthogonal tiling of 2-D iterations. Journal of Parallel and Distributed Computing.(1997) 45:159–165
6. Andonov, R., Bourzoufi, H., Rajopadhye, S.: Two dimensional orthogonal tiling: from theory to practice. International Conference on high Performance Computing (HiPC).(1996) 225–2 31. India, IEEE Computer Society Press.
7. Andonov, R., Yanev, N., Bourzoufi, H.: Three-dimensional orthogonal tile size: Mathematical programming approach. International Conference on Application Specific Systems, Architectures and Processors (ASAP).(1997) 209–218. Zurich, IEEE Computer Society.
8. Palermo, D., Su, E., Chandy, A., Banerjee, P.: Communication Optimization Used in the PARADIGM Compiler for Distributed Memory Multicomputers. International Conference on Parallel Processing.(1994).
9. Andonov, R., Rajopadhye, S., Yanev, N.: Optimal Orthogonal Tile. in Europar'98, pages 480-490, September 1998.
10. Weingartner, H.M. Ness, D.N.: Methods for the solution of the Multidimensional 0–1 Knapsack problem.” Operations Research. (1967)15(1), 83–103.

Session II-B

Scheduling

Chair: Rajib Mall

Indian Institute of Technology, Kharagpur

Process Migration Effects on Memory Performance of Multiprocessor Web-Servers

Pierfrancesco Foglia, Roberto Giorgi, and Cosimo Antonio Prete

Dipartimento di Ingegneria della Informazione
Università' di Pisa
Via Diotisalvi, 2 – 56100 Pisa, Italy
`{foglia,giorgi,prete}@iet.unipi.it`

Abstract. In this work we put into evidence how the memory performance of a Web-Server machine may depend on the sharing induced by process migration. We considered a shared-bus shared-memory multiprocessor as the simplest multiprocessor architecture to be used for accelerating Web-based and commercial applications. It is well known that, in this kind of system, the bus is the critical element that may limit the scalability of the machine. Nevertheless, many factors that influence bus utilization, when process migration is permitted, have not been thoroughly investigated yet. We analyze a basic four-processor and a high-performance sixteen-processor machine. We show that, even in the four-processor case, the overhead induced by the sharing of private data as a consequence of process migration, namely passive sharing, cannot be neglected. Then, we consider the sixteen-processor case, where the effects on performance are more massive. The results show that even though the performance may take advantage of larger caches or from cache affinity scheduling, there is still a great amount of passive sharing, besides false sharing and active sharing. In order to limit false sharing overhead, we can adopt an accurate design of kernel data structures. Passive sharing can be reduced, or even eliminated, by using appropriate coherence protocols. The evaluation of two of such protocols (AMSD and PSCR) shows that we can achieve better processor utilization compared to the MESI case.

1 Introduction

Multiprocessor systems are becoming more and more widespread due to both the cheaper cost of necessary components and the advances of technology. In their simplest implementation, multiprocessors are built by using commodity processors, shared memory, and shared bus, in order to achieve high performance at minimum cost [10]. Their relatively high diffusion made it common to find these machines running not only traditional scientific applications, but also commercial ones, as is the case for Web-Server systems [2],[3].

The design of such systems is object of an increasing interest. Nevertheless, we are not aware of papers that thoroughly investigate some factors that can heavily influence their performance, e.g. the effect of process migration, and the consequent passive-sharing [16]. If those aspects were considered correctly, different architectural choices would be dictated to tune the performance of Web-Server multiprocessors.

Web-Server performance is influenced by network features, I/O subsystem performance (to access database information or HTML pages) and by hardware and software architecture both from client and server side. In the following, we shall

consider a server based on shared-bus shared-memory multiprocessor, and in particular, we shall focus on the core architecture related problems, and the effects of process migration.

The design issues of a multiprocessor system are scalability and speedup. These goals can be achieved by using cache memories, in order to hide the memory latency, and reduce the bus traffic (the main causes that limit speed up and scalability). Multiple cache memories introduce the coherence problem and the need of a certain number of bus transactions (known as *coherence overhead*) that add to the basic bus traffic of cache-based uniprocessors. Thus, a design issue is also the minimization of such coherence overhead. The industry-standard MESI protocol may not be enough effective to minimize that overhead, in the case of Web-Servers.

The scheduling algorithm plays an essential role in operating systems for multiprocessors because it has to obtain the load balancing among processors. The consequent process migration generates passive sharing: private data blocks of a process can become resident in multiple caches and generate useless coherence-related overhead, which in turn may limit system performance [16].

In this paper, we shall analyze the memory hierarchy behavior and the bus coherence overhead of a multiprocessor Web-Server. The simulation methodology relies on trace-driven technique, by means of *Trace Factory* environment [8], [15].

The analysis starts from a reference case, and explores different architectural choices for cache and number of processors. The scheduling algorithm has also been varied, considering both a random and a cache affinity policy [18]. The results we obtained show that, in these systems, large caches and cache affinity improve the performance. Anyway, due to both false sharing and passive sharing, MESI does not allow achieving the best performance for Web-Server applications. Whilst a re-design of operating system kernel structures is suggested to reduce false sharing, passive sharing can be eliminated by using new coherence protocol schemes. We evaluated two different solutions for the coherence protocol (AMSD [19], [4] and PSCR [9]). The results show a sensible performance increase, especially for high performance architectures.

2 Coherence Overhead

In a shared-bus, shared-memory multiprocessors, speed-up and scalability are the design issues. It is well known that the shared bus is the performance bottleneck, in this kind of systems. To overcome the bus limitations, and achieving the design goals, we need to carefully design the memory subsystem. These systems usually include large cache memories that contribute in both hiding memory latency, and reducing the bus traffic [10], but they generate the need for a coherence protocol [14], [22], [23]. The protocol activity involves a certain number of bus transactions, which adds to the basic bus traffic of cache-based uniprocessors, thus limiting the system scalability.

In our evaluations, we have considered the MESI protocol. MESI is a Write-Invalidate protocol [21], and it is used in most of the actual high-performance microprocessors, like the AMD K5 and K6, the PowerPC series, the SUN UltraSparc II, the SGI R10000, the Intel Pentium, Pentium Pro, Pentium II and Merced. The remote invalidation used by MESI (*write-for-invalidate*) to obtain coherency has as a drawback the need to reload the copy, if it is used again by the remote processor, thus

generating a miss (*Invalidation Miss*). Therefore, MESI coherence overhead (that is the transactions needed to enforce coherence) is due to *Invalidation Misses* and *write-for-invalidate* transactions.

We wish to relate that overhead with the kinds of data sharing that can be observed in the system, in order to detect the causes for the coherence overhead, and thus finding out the appropriate hardware/software solutions. Three different types of data sharing can be observed: i) *active sharing*, which occurs when the same cached data item is referenced by processes running on different processors; ii) *false sharing* [25], which occurs when several processors reference different data items belonging to the same memory block; iii) *passive* [23], [16] or *process-migration* [1] *sharing*, which occurs when a memory block, though belonging to a private area of a process, is replicated in more than one cache as a consequence of the migration of the owner process. Whilst active sharing is unavoidable, the other two forms of sharing are useless. The relevant overhead they produce can be minimized [24], [19], [13], and possibly avoided [9].

3 Workload Description

The Web-Server activity is reproduced by means of the Apache daemon [17], which handles HTTP requests, an SQL server, namely PostgreSQL [29], which handles TPC-D [26] queries, and several Unix utilities, which interface the various programs.

Table 1. Statistics of source traces for some UNIX utilities (64-byte block size and 5,000,000 references per application)

APPLICATION	Distinct blocks	Code (%)	Data Read	Data Write
AWK (BEG)	4963	76.76	14.76	8.47
AWK (MID)	3832	76.59	14.48	8.93
CP	2615	77.53	13.87	8.60
GZIP	3518	82.84	14.88	2.28
RM	1314	86.39	11.51	2.10
LS -AR	2911	80.62	13.84	5.54
LS -LTR (BEG)	2798	78.77	14.58	6.64
LS -LTR (MID)	2436	78.42	14.07	7.51

Web-Server software relies on server-client paradigm. The client usually requests HTML files or Java Applets by means of a Web browser. The requested file is provided and sent back to the client side. We used the Apache daemon [5]. We configured the daemon, so that it can spawn a minimum of 2 idle processes and a maximum of 10 idle processes. A child processes up to 100 requests before dying.

Besides this file transferring activity, Web-Servers might need to browse their internal database [3]. The workload thus includes a search engine, namely PostgreSQL DBMS [29]. We instructed our Web-Server to generate some queries, upon incoming calls, as specified by the TPC-D benchmark for Decision Support Systems (DSS). PostgreSQL consists of a front-end process that accepts SQL queries, and a backend that forks processes, which manage the queries. A description of PostgreSQL memory and synchronization management can be found in [27]. TPC-D simulates an application for a wholesale supplier that manages, sells, and distributes a product worldwide. It includes 17 read-only queries, and 2 update queries. Most of

the queries involves a high number of records, and perform different operations on database tables.

For completing our Web-Server workload, we considered some glue-processes that can be generated by shell scripts (`ls`, `awk`, `cp`, `gzip`, and `rm`). In a typical situation, various requests may be running, thus requiring the support of different system commands and ordinary applications.

Table 2. Statistics of multiprocess application source traces (Apache and TPC-D) and our target trace (SWEB), in case of 64-byte block size and 5,000,000 references per process.

Workload	Number of processes	Distinct Blocks	Code (%)	Data (%)		Shared blocks	Shared data (%)	
				Read	Write		Accesses	Write
APACHE	13	95343	75.14	18.24	6.61	666	1.52	0.49
TPC-D	5	8821	71.94	18.17	9.88	2573	2.7	0.79
SWEB	26	239848	75.49	17.12	7.39	3982	1.68	0.54

4 Performance Analysis

4.1 Methodology

The methodology used in our analysis is based on trace-driven simulation [20], [15], [28], and on the simulation of the three kernel activities that most affect performance: *system calls*, *process scheduling*, and *virtual-to-physical address translation* [7]. In the first phase, we produce a *source* trace (a sequence of user memory references, system-call positions, and synchronization events in case of multithreaded programs) for each application belonging to the workload, by means of a tracing tool. In the second phase, Trace Factory simulates the execution of complex workloads by combining multiple source traces, generating the references of system calls, and by simulating process scheduling and virtual-to-physical translation. Trace Factory furnishes the references (*target* trace) to a memory-hierarchy simulator [15], by using an on-demand policy. Indeed, Trace Factory produces a new reference whenever the simulator requests one, so that the timing behavior imposed by the memory subsystem conditions the reference production [8].

To detect sharing patterns, and evaluate the source of overhead, we have extended an existing classification algorithm [11] to the case of passive sharing, and the case of finite-size caches.

The Web-Server workload is constituted by 26 processes, 13 of which are spawned by the Apache daemon, 5 by PostgreSQL (corresponding to the first 5 queries of the TPC-D benchmark), and 8 processes are Unix utilities. Table 1 (for the uniprocess applications) and 2 (for the multiprocess ones) contain some statistics of the source traces that we used to generate the workload. To take into account that some requests may be using the same program at different times, we traced some commands in shifted execution sections: initial (beg) and middle (mid). Table 2 contains also the statistics of the target workload that we used in our evaluation (SWE). Data are related to 100 requests to the Web-Server, that produce 130 millions of references.

4.2 Simulation Results

As base case-study, a machine with 128-bit shared bus is considered. As for the number of processors, two configurations have been studied: a basic machine with 4 processors and a high-performance machine with 16 processors. For the scheduling policy, two solutions have been considered: *random* and *cache-affinity*; scheduler time slice is 200,000 references. Cache size has been varied between 32K and 2M. The simulated processors are MIPS-R10000-like; paging relays on 4-KByte-page size; the bus logic supports transaction splitting, and processor-consistency memory model [6]. The base case study timings for the simulator are summarized in Table 3.

Table 3. Numerical values of timing parameters for the multiprocessor simulator (times are in clock cycles) in the case of 64-byte block size.

CLASS	PARAMETER	TIMINGS
CPU	READ/WRITE CYCLE	2
BUS	WRITE FOR INVALIDATE TRANSACTION	5
	MEMORY-TO-CACHE READ-BLOCK TRANSACTION	72
	CACHE-TO-CACHE READ-BLOCK TRANSACTION	16
	UPDATE-BLOCK TRANSACTION	10

This paper is mainly concerned in evaluating the effects of process migration on the performance. The performance is affected by the cache misses (which essentially contribute to determine the mean memory access time) and the bus-traffic (which affects the miss cost). In the case of MESI protocol, bus traffic is due to the following transactions: *read-block* transactions (essentially due to the misses), *write-for-invalidate* transactions and *update* transactions. *Update* transactions are only a neglectable part of the bus-traffic and they do not influence greatly our analysis. The rest of the traffic is due to classical misses (sum of cold and replacement misses) and coherence traffic, constituted of *Invalidation Misses* and *write-for-invalidate* transactions.

Process migration affects the miss rate, since a migrating process has to regenerate its working set on a new processor, while it is destroying the one of the other processes. Moreover, process migration affects the coherence traffic, since it generates passive sharing: private data blocks of a process can become resident in multiple cache and generate useless coherence-related overhead [16]. Thus, our evaluation includes the analysis of both the miss rate and the coherence traffic.

We first analyze the miss rate (Figure 1) when cache size and number of ways are varied in the case of four processors. As expected, invalidation miss (i.e. true and false sharing miss, both due to kernel and user) increases with larger cache sizes. Nevertheless, the total miss rate decreases significantly as the cache size is increased up to 2M bytes, and in the case of more ways. For cache sizes above 512K bytes the difference between 2 and 4 ways becomes neglectable. In Figure 2, we detail the invalidation misses of Figure 1. Invalidations misses are basically due to the kernel, and in that case, false sharing is the main source of this overhead.

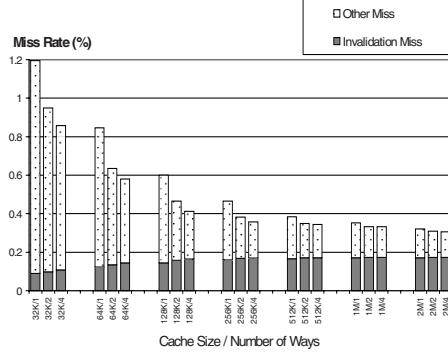


Fig. 1. Miss rate versus cache size (32K bytes, 64K bytes, 128K bytes, 256K bytes, 512K bytes, 1M bytes, 2M bytes), and number of ways (1, 2, 4). In this experiment, we had a 4-processors configuration, and random scheduling policy. "Other Miss" includes cold miss, capacity miss and replacement miss. Miss rate decreases while invalidation miss (i.e. the sum of false sharing miss and true sharing miss) increases with large cache size and more associativity.

False sharing occurs when data are used in exclusive way by different processes, and those data become physically shared. This happens when data are improperly aligned, i.e. when different data objects are placed into the same block. False sharing can be eliminated either by using special coherence protocols [24], or properly allocating the involved shared data structures [12]. This latter solution seems more suitable for our case study, since the detected false sharing is mainly due to kernel, which is a completely known part of the system at design time. Invalidation miss rate is significant for large cache sizes, whilst is neglectable for small cache sizes. Therefore, for cache sizes above 256K bytes, our results indicate that the kernel of Web-Server should be designed according to the above techniques, in order to avoid false sharing effects.

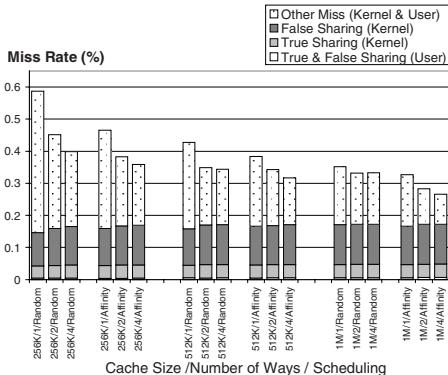


Fig. 3. Breakdown of miss rate versus cache sizes (256K bytes, 512K bytes, 1M bytes), number of ways (1, 2, 4), and scheduling policy (random, affinity), for a 4-processor configuration. Invalidation misses, and in particular, false sharing misses, increase in the 16-processor configuration.

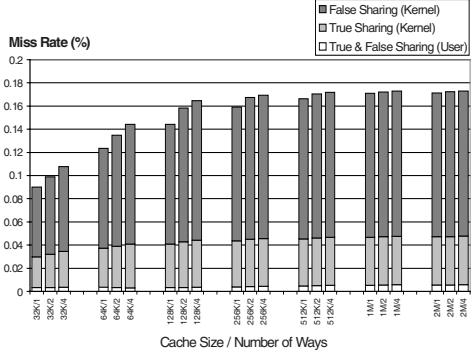


Fig. 2. Detail of the invalidation miss versus cache size (32K bytes, 64K bytes, 128K bytes, 256K bytes, 512K bytes, 1M bytes, 2M bytes), and number of ways (1, 2, 4), for a 4-processor configuration with a random scheduling policy. Invalidation misses are basically due to the kernel and false sharing is the main source of overhead.

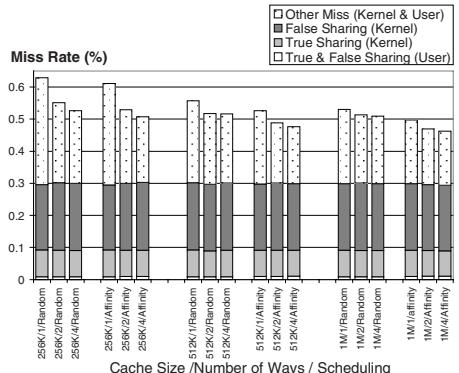


Fig. 4. Breakdown of miss rate versus cache sizes (256K bytes, 512K bytes, 1M bytes), number of ways (1, 2, 4), and scheduling policy (random, affinity), for the 16-processor configuration.

To highlight the effects of process migration on miss rate we varied the scheduling policy (random and cache affinity). Figures 3 and 4 investigate the effects in the case of 4 processor and 16 processors, respectively. Cache affinity mainly causes a reduction of the "other misses". In the 4-processor case the benefit is higher than in the 16-processor case, due to a larger number of processes compared to the number of processors. Indeed, in this condition, there is a larger number of ready-to-execute processes, so that we have a high probability that a process can execute its time-slice on the last-assigned processor. Thus, cache affinity seems to be not so effective for high-end machines. Comparing Figures 3 and 4, we also notice that in the case of 16 processors the miss rate is higher, as we expected, due to the presence of more copies of the same block. This is caused by the larger number of processors (and caches).

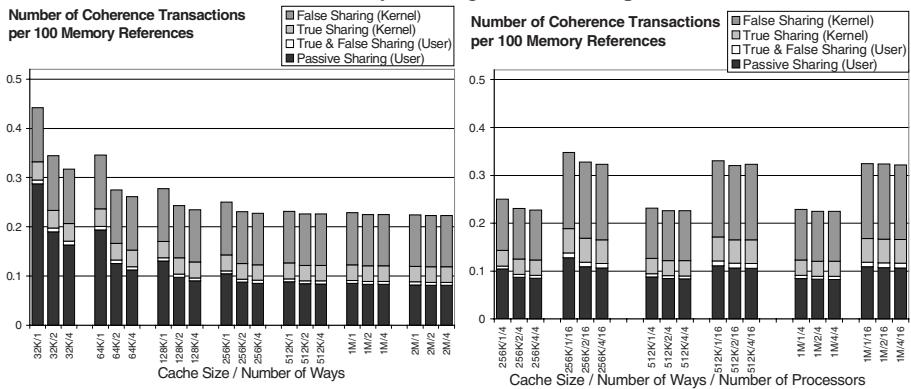


Fig. 5. Number of coherence transactions of MESI versus cache size (32K bytes, 64K bytes, 128K bytes, 256K bytes, 512K bytes, 1M bytes, 2M bytes) and number of ways (1, 2, 4), for 4-processor configuration and a random scheduling policy. The decrease of passive sharing transactions with the associativity is due to the reduction of passively shared copies produced by the write-for-invalidate transactions. That effect is more evident for small cache sizes.

As observed above, cache misses produce a relevant traffic of *read-block* transactions on the bus. Thus, the graphs in Figures 1, 3, 4, can be read as the *number of read-block transactions per 100 memory references*. The most significant part of the rest of transactions is due to coherence transactions (e.g. *write-for-invalidate* transactions, in the case of MESI). Figure 5 and 6 show such coherence transactions when cache size, ways, and number of processors is varied. In order to allow the comparison between *read-block* and *write-for-invalidate* traffic, we used the same metric for the two quantities: *number of bus transactions generated by 100 references*.

As for the overhead produced by shared data accesses, the behavior seen in the previous figures is confirmed: kernel related overhead is more consistent than user-related overhead and false sharing dominates it. However, user accesses also exhibit a noticeable amount of passive sharing, produced by private data as a consequence of process migration. Passive sharing overhead decreases with larger cache sizes and with a higher number of ways. This non-intuitive result is more evident in the case of small caches, and is a consequence of two competing factors. On one hand, there is an increase of passive shared copies due to their longer lifetime. This is produced by

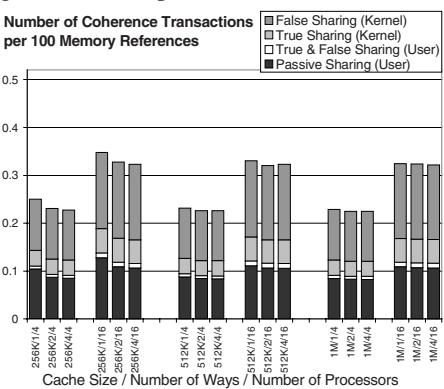


Fig. 6. Number of coherence transactions (write-for-invalidate transactions) versus cache size (256K bytes, 512K bytes, 1M bytes), number of processors (4, 16), and number of ways (1, 2, 4), for a random scheduling policy. There is an increase in each component of this overhead in the high-end (16 processor) configuration.

increase of cache size, or set-associativity. On the other, MESI invalidates remote copies on every write-miss, thus reducing the passively shared copies.

Passive sharing increases when switching from the 4- to the 16-processor configuration (Figure 6). As it happens for false sharing, also passive sharing is a "useless" sharing, since it is consequence of migration and it is not caused by accesses to active shared data.

We considered three different techniques that could reduce passive sharing: two are based on coherence protocol (PSCR [9], and AMSD [19], [4]), and one is based on an affinity scheduling algorithm [18]. AMSD is our acronym, which stands for Adaptive Migratory Sharing Detection. Migratory sharing is characterized by the exclusive use of data for a long time interval. Typically, the control over these data migrates from one process to another. The protocol identifies migratory-shared data dynamically in order to reduce the cost of moving them. The implementation relies on an extension of a common MESI protocol. Coherence traffic is again due to *write-for-invalidate* transactions and *invalidation* transactions.

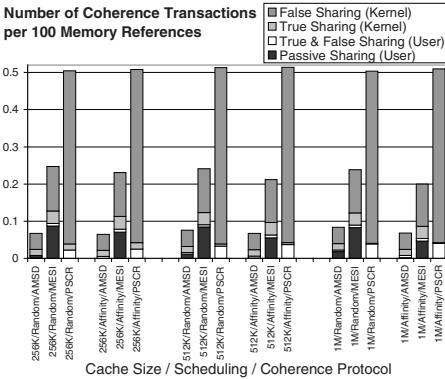


Fig. 7. Number of coherence transactions of versus cache size (256K bytes, 512K bytes, 1M bytes), scheduling algorithm (random, affinity), coherence protocol (MESI, PSCR, AMSD). Data assume 4 processors and a two-way set associative cache. Both AMSD and PSCR are effective in reducing or eliminating passive sharing overhead.

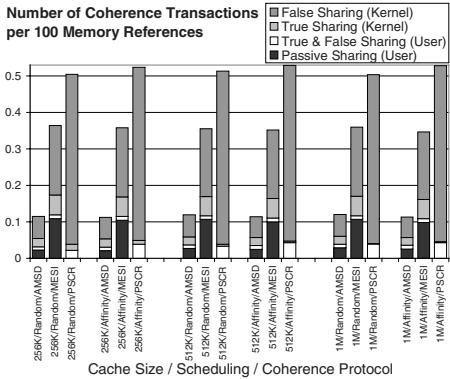


Fig. 8. Number of coherence transactions versus cache size (256K bytes, 512K bytes, 1M bytes), scheduling algorithm (random, affinity), and coherence protocol (AMSD, PSCR, MESI). Data assume 16 processors and a two-way set associative cache. The affinity scheduling produces only a slightly reduction of passive sharing.

PSCR (Passive Shared Copy Removal) adopts a selective invalidation for the private data, and uses the *write-update* scheme for the actively shared data. A cached copy belonging to a process private area is invalidated locally as soon as another processor fetches the same block.

In the case of four processors (Figure 7), the adoption of AMSD causes a reduction of passive, true, and false sharing transactions compared to the MESI case. PSCR eliminates the transactions due to passive sharing, but causes a higher number of them in the other kinds of sharing, due to the broadcast nature of the protocol. Cache affinity technique also reduces passive sharing transactions, so it does not improve PSCR performance. In the case of 16 processors (Figure 8), we have the same effects for the number of transactions, except in the case of affinity. Indeed, since the number of processors is getting closer to the number of processes, the affinity scheduling is not always applicable. The scheduler is forced to allocate a ready process on a different available processor.

Figure 9 also considers the cost of transactions combining the effects on performance in a single figure, the Global System Power (i.e. the sum of processor utilization [9]). Again, we varied the cache size, the protocol, the scheduling policy, and the number of processors. In the case of four processors, all the three techniques do not produce a sensible effect on performance. The situation is very different in the case of sixteen processors. In that case PSCR and AMSD allow much better performance than MESI. As shown in Figure 10 (and 8) this is due to a reduction of the miss rate for PSCR and to a reduction of coherence transactions for AMSD.

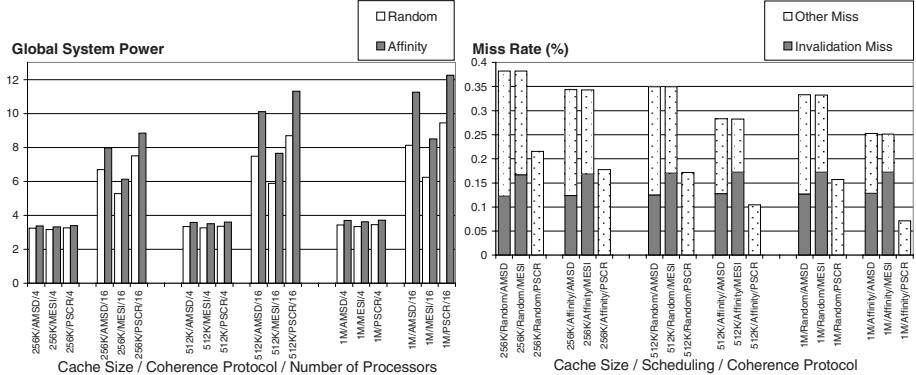


Fig. 9. Global System Power versus cache size (256K bytes, 512K bytes, 1M bytes), coherence protocol (AMSD, PSCR, MESI), and number of processors (4, 16) for two scheduling policy (random, affinity). Data assume a two way set associative cache. Global System Power is the sum of processor utilizations. In all cases PSCR and AMSD show better processor utilization than MESI.

Fig. 10. Miss Rate versus cache size (256K bytes, 512K bytes, 1M bytes), scheduling policy (random, affinity), and coherence protocol (AMSD, PSCR, MESI). Data assume a 4-processor configuration, and a two way set associative cache. The miss rate is almost the same for MESI and AMSD, even if their miss component actually varies. PSCR allows a much lower miss rate compared to the other protocols' due to the absence of the invalidation miss.

5 Conclusions

In this paper we characterized the number and type of transactions induced on the shared-bus of a shared-memory multiprocessors used as a Web-Server machine. Our workload has been set up by tracing a combined workload made of an HTTP server (Apache), PostgreSQL DB-server resolving TPC-D benchmark queries, and typical UNIX shell commands. The analysis has been carried out with trace-driven simulation, and by considering not only user references, but also the most influencing kernel activities. The low usage of shared memory in the user space causes most of the data sharing to happen in the kernel space. The larger part of this sharing has resulted to be due to false sharing, and secondly to passive sharing. To eliminate false sharing, we suggest the redesign of kernel structures. We put into evidence how the MESI protocol is not capable of treating passive sharing, that is the sharing generated on private data as a consequence of process migration. These facts indicate that margins to improve MESI protocol still exist. The use of affinity scheduling algorithms determines only a partial reduction of passive sharing and this technique does not adapt to all load conditions. Ad hoc protocols, like PSCR and AMSD, better reduce the main causes of overhead and allow better performance than MESI.

References

1. A. Agarwal and A. Gupta, "Memory Reference Characteristics of Multiprocessor Applications under Mach". *Proc. ACM Sigmetrics*, Santa Fe, NM, pp. 215-225, May 1998.
2. M. Baentsch, L. Baum, and G. Molter, "Enhancing the Web's Infrastructure: From Caching to Replication". *Internet Computing*, vol. 1, no. 2, pp. 18-27, Mar.-Apr. 1997.
3. L. A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads". *Proc. of the 25th Intl. Symp. on Computer Architecture*, pp. 3-14, June 1998.
4. A. L. Cox and R. J. Fowler, "Adaptive Cache Coherency for Detecting Migratory Shared Data," *Proc. 20th Intl. Symp. on Computer Architecture*, San Diego, California, pp. 98-108, May 1993.
5. J. Edwards, "The changing Face of Freeware". *IEEE Computer*, Vol. 31, N. 10, pp 11-13, Oct. 1998.
6. K. Gharachorloo, A. Gupta, and J. Hennessy, "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors", *Proc. of the 4th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, California, pp. 245-357, Apr. 1991.
7. R. Giorgi, C. Prete, G. Prina, L. Ricciardi, "A Hybrid Approach to Trace Generation for Performance Evaluation of Shared-Bus Multiprocessors". *Proc. 22nd EuroMicro Intl. Conf.*, Prague, pp. 207-241, September 1996.
8. R. Giorgi, C. A. Prete, G. Prina, L. Ricciardi, "Trace Factory: a Workload Generation Environment for Trace-Driven Simulation of Shared-Bus Multiprocessor". *IEEE Concurrency*, 5(4), pp. 54-68, Oct-Dec 1997.
9. R. Giorgi, C. A. Prete, "PSCR: A Coherence Protocol for Eliminating Passive Sharing in Shared-Bus Shared-Memory Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, n. 7, pp. 742-763, July 1999.
10. J. Hennessy and D.A. Patterson, *Computer Architecture: a Quantitative Approach*, 2nd edition. Morgan Kaufmann Publishers, San Francisco, CA, 1996.
11. R. L. Hyde and B. D. Fleisch, "An Analysis of Degenerate Sharing and False Coherence". *Journal of Parallel and Distributed Computing*, 34(2), pp. 183-195, May 1996.
12. T. E. Jeremiassen and S. J. Eggers, "Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations", *ACM SIGPLAN Notice*, 30(8), pp.179-188, Aug. 1995.
13. C. A. Prete, "A new solution of coherence protocol for tightly coupled multiprocessor systems," *Microprocessing and Microprogramming*, vol. 30, no. 1-5, pp. 207-214, 1990.
14. C. A. Prete, "RST Cache Memory Design for a Tightly Coupled Multiprocessor System," *IEEE Micro*, vol. 11, no. 2, pp. 16-19, 40-52, Apr. 1991.
15. C. A. Prete, G. Prina, and L. Ricciardi, "A Trace Driven Simulator for Performance Evaluation of Cache-Based Multiprocessor System". *IEEE Transactions on Parallel and Distributed Systems*, vol. 6 (9), pp. 915-929, September 1995
16. C. A. Prete, G. Prina, R. Giorgi, and L. Ricciardi, "Some Considerations About Passive Sharing in Shared-Memory Multiprocessors". *IEEE TCCA Newsletter*, pp.34-40, Mar. 1997.
17. D. Robinson and the Apache Group, APACHE - An HTTP Server, Reference Manual, 1995. .
18. M. S. Squillante, D. E. Lazowska, "Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling". *IEEE Transactions on Parallel and Distributed Systems*, vol. 4 (2), pp. 131-143, February 1993.
19. P. Stenstrom, M. Brorsson, and L. Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing". *Proc. of the 20th Intl. Symp. on Computer Architecture*. San Diego, CA, May 1993.
20. C. B. Stunkel, B. Janssens, and W. K. Fuchs, "Address Tracing for Parallel Machines," *IEEE Computer*, vol. 24, no. 1, pp. 31-45, Jan. 1991.
21. P. Sweazey and A.J. Smith, "A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus". *Proc. of the 13th Intl. Symp. on Computer Architecture*, pp. 414-423, June 1986.
22. M. Tomasevic and V. Milutinovic, *The Cache Coherence Problem in Shared-Memory Multiprocessors -Hardware Solutions*. IEEE Computer Society Press, Los Alamitos, CA, April 1993.
23. M. Tomasevic and V. Milutinovic, "Hardware Approaches to Cache Coherence in Shared-Memory Multiprocessors". *IEEE Micro*, vol. 14, no. 5, pp. 52-59, Oct. 1994 and vol. 14, no. 6, 61-66, Dec. 1994.
24. M. Tomasevic and V. Milutinovic, "The word-invalidate cache coherence protocol", *Microprocessors and Microsystems*, pp. 3-16, vol. 20, Mar. 1996.
25. J. Torrellas, M. S. Lam, and J.L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches". *IEEE Transactions on Computer*, vol. 43, n. 6, pp. 651-663, June 1994.
26. Transaction Processing Performance Council, TPC Benchmark D Standard Specification. Dec 1995.
27. P. Trancoso, J. L. Larriba-Pey, Z. Zhang, and J. Torrellas, "The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors". *Proc. of the 3rd Intl. Symp. on High Performance Computer Architecture*, Feb 1997.
28. R. A. Uhlig and T. N. Mudge, "Trace-Driven Memory Simulation: a survey". *ACM Computing Surveys*, pp. 128-170, June 1997.
29. A. Yu and J. Chen, The POSTGRES95 User Manual. Computer science Div., dept of EECS, University of California at Berkeley, July 1995.

Adaptive Algorithms For Scheduling Static Task Graphs In Dynamic Distributed Systems

Prashanti Das¹, Dibyendu Das² and Pallab Dasgupta³

¹ Motorola India Electronics Ltd.

² Hewlett Packard, India Software Operation Pvt. Ltd.

³ Dept. of Computer Sc. and Engg, I.I.T Kharagpur, India

Abstract. In this paper we consider the problem of scheduling a given task graph on a dynamic network, where processors may become available or unavailable during the lifetime of the computation. We show that known list scheduling algorithms which use task cloning can be extended to develop efficient algorithms in this model. We also present a different approach where in anticipation of processor failures and recoveries, a set of schedules are precomputed and schedule switching is done whenever a failure or recovery takes place.

1 Introduction

Recent research on task scheduling with cloning has shown that cloning or duplicating tasks can significantly reduce the length of the overall schedule [2, 3, 4, 5]. While the task of finding an optimal schedule using cloning has been shown to be NP-hard, efficient polynomial time algorithms for task scheduling with cloning have been developed which give a good approximation of the optimal schedule [3, 4]. However, these algorithms assume that there is no bound on the number of processors. It has been shown that determining the processor requirement for optimally task scheduling with cloning is NP-hard [1]. Efficient heuristic based list scheduling algorithms have also been developed for task scheduling (with cloning) on a *given set of processors* [2, 5].

In this paper we address the following problem. We are given a directed acyclic task graph with known task execution times (represented as weights of the nodes of the task graph) and known data transfer times between pairs of tasks (represented as edge weights) if scheduled in different processors. The initial set of available processors is also given. With time, the set of available processors may change, that is, some processors may *fail* (become unavailable) and some processors may *recover* (become available). The set of available processors at a given instant of time becomes known only at that instant of time. We require fault tolerant strategies which reschedule and reallocate tasks dynamically and attempt to complete the computation represented by the given task graph at the earliest.

The contributions of this paper are as follows:

1. We extend the known heuristic based scheduling algorithms, ISH and DSH [5], to this model and show that the DSH-extension which uses cloning is well suited for this model.

2. We propose two strategies which pre-compute a set of *non-dominated* schedules, and perform a dynamic schedule mapping at runtime whenever a failure or recovery takes place. Experimental results show that the schedule lengths compare well with the DSH-extension, and therefore runs faster, since the scheduling overhead is minimized.

2 Problem Definition and Preliminaries

The algorithms presented in this paper are based on the following model. At a given instance of time, τ , the distributed system consists of a set of homogeneous processors, $S^\tau = \{P_1^\tau, \dots, P_k^\tau\}$. It is assumed that the distributed system is fully connected, that is, there is a communication channel between every pair of processors. Throughout this paper we ignore the contention on communication channels. The task scheduling problem in this model is defined as follows:

Given: A directed acyclic task graph consisting of the following:

1. A set of nodes representing a set of tasks $T = \{t_1, \dots, t_n\}$ to be executed.
2. A partial order, \prec , defined on T which specifies precedence between tasks. $t_i \prec t_j$ signifies that t_i must be completed before t_j can begin, that is, t_j requires some data from t_i . The given task graph has an edge (t_i, t_j) iff $t_i \prec t_j$.
3. For each task t_i , we are given its execution time, w_i . Since the system consists of similar processors, w_i is independent of the processor in which it is executed.
4. For each edge (t_i, t_j) in the task graph, we are given an edge weight, c_{ij} , which represents the time required to transfer data from t_i to t_j provided they are scheduled in different processors. c_{ij} is proportional to the volume of data transferred from t_i to t_j .

To find: The shortest length schedule for the set of tasks T on the set, S^τ , of available processors at time τ .

The model for communication delays is as follows. We assume the existence of an independent communication interface for each pair of processors, that is, a processor can execute a task and communicate with one or more processors at the same time. In such a model, for any two tasks t_i and t_j , if $t_i \prec t_j$ and t_i has been scheduled in processor P at time τ , then t_j should be scheduled either on processor P at time $\tau' > \tau + w_i$ or on some other processor at time $\tau' > \tau + w_i + c_{ij}$. We make a few simplifying assumptions about the system:

- As soon as the set of available processors change, it becomes universally known and the rescheduling process can start without any delay.
- Each failure and each recovery occur in distinct instances of time.
- Each task, on completion, writes the data required by its successors on a shared (broadcast) medium. This is quite reasonable, since a shared filesystem (which acts as the broadcast medium) is common in clusters.

Based on these assumptions, we introduce the definitions of *edge dropping* and *node dropping*.

Definition 1. [Edge Dropping:]

An edge (t_i, t_j) in a task graph is said to be dropped iff either of the following hold:

1. The processor where t_i is scheduled remains available up to a time, $\tau + w_i + c_{ij}$, from the time τ when t_i starts executing in that processor.
2. The node representing the task t_j is “dropped” as per Definition 2. \square

Definition 2. [Node Dropping:]

A node in the task graph representing a task, t_i , is said to be dropped at time τ if all its outgoing edges in the task graph are dropped by time τ . \square

The assumptions made in our model lead to the following lemma.

Lemma 3. *The task represented by a dropped node is globally completed, that is, it never needs to be rescheduled even if the processor where it executed becomes unavailable.*

Proof: Consider an outgoing edge (t_i, t_j) of a node t_i dropped at time τ . The edge may be dropped due to the satisfaction of either of the two conditions specified in Definition 1. If the edge is dropped by the first condition, then all processors have access to the data required from t_i for executing t_j , and therefore it will never be necessary to re-execute t_i for this data. If the edge is dropped by the second condition, then by induction, if we are given that the task t_j never needs to be re-executed then it follows that the task t_i never needs to be re-executed for the data required by t_j . It follows that if all edges of a task are dropped, then the task never needs to be re-executed. \square

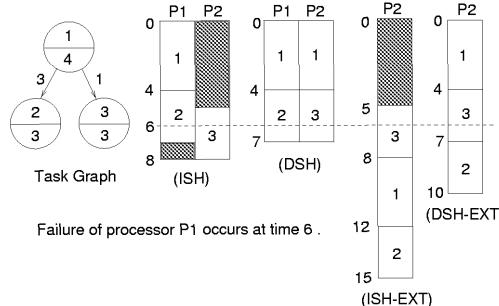


Fig. 1. Example showing the benefit of task cloning

The following example, illustrates the benefits of task cloning.

Example 1. Figure 1 shows a simple task graph with 3 nodes. The task numbers and their execution times are indicated on the upper and lower halves of the nodes respectively. The data transfer times are shown on the edges. Figure 1 also shows the task schedules (as GANT charts) using ISH (no cloning) and DSH (with cloning), as well as ISH-extension and DSH-extension after processor p_1 fails at time step 6. By virtue of cloning task 1, not only is the DSH schedule shorter than the ISH schedule, but the DSH extension is also able to complete earlier than the ISH extension. \square

2.1 Heuristics for scheduling

In this subsection, we describe in brief two very well known heuristics [5] for list scheduling which have been used in this work. As in all list scheduling algorithms, both these methods select tasks one by one, based on the partial order, \prec , that is, a task t_i is selected only when all its predecessors have been scheduled. A selected task is scheduled in the processor where it can start at the earliest. The following heuristics differ in their strategies for finding the appropriate processor for each selected task.

Insertion Scheduling Heuristic (ISH): In this method, the earliest starting time at each processor for the selected task is compared and the task is scheduled in the processor which offers the minimum start time. No task cloning is considered.

Duplication Scheduling Heuristic (DSH): In this method, the earliest start time of the selected task at a given processor is determined by considering the possibilities of cloning its predecessors into that processor. The selected task is scheduled in the processor which offers the minimum start time.

3 Extensions of DSH and ISH

In this section, we describe in brief the extensions of ISH and DSH to our model and present results which show that the DSH-extension is more useful than the ISH-extension in our model.

The extensions of ISH and DSH start by computing a schedule of the given task graph on the initial set of processors. Whenever a processor failure or recovery occurs, a rescheduling operation is carried out taking into account the tasks which have already completed. Figure 2 shows the steps in reconstructing the partial task graph, creating the intermediate set of ready tasks, and the modification of the GANT chart. After this construction, the ISH extension uses ISH while the DSH extension uses DSH for rescheduling.

The ISH and DSH extensions were executed on the randomly generated sample graphs of Table 1. For the graphs with 15 nodes, we took the initial number of available processors as 4, and for all the other graphs (namely, 50 and 100 node graphs), we took the initial number of available processors as 10. We executed the algorithms subject to single and multiple failures, as well as recovery.

Figure 3-7 show the comparisons between the ISH-extension and DSH-extension in terms of the percentage performance degradation with respect to the DSH schedule lengths. By percentage degradation we mean 100 times the difference in the overall schedule length with the DSH schedule length, divided by the DSH schedule length. Figure 3, Figure 4 and Figure 5 show the comparison when a failure occurs respectively at the 10%, 50% and 90% time of the initial DSH schedules. The processor whose failure causes maximum degradation is assumed to fail. Figure 6 shows the comparisons when two failures occur, respectively at the 10% and 50% times. Figure 7 shows the comparison when failures occur at the 10% and 50% times and a recovery occurs at the 60% time. The results

Re-constructing the partial task graph at time τ :

1. Compute the set of dropped nodes and edges at time τ .
2. Remove from the original task graph the set of dropped nodes and edges.
3. In the new task graph, mark the tasks which have completed or are executing at time τ in the remaining available processors as *DONE*.

Creating the list of ready tasks at time τ :

Create a list of ready tasks comprising of the following nodes:

1. Unmarked nodes which have no predecessor
2. Unmarked nodes all of whose predecessors are marked *DONE*.

Creating the modified GANT chart:

1. In the GANT chart of each processor, fill up all holes upto time τ by dummy tasks to prevent DSH/ISH from scheduling anything there
2. Remove from the GANT charts all tasks scheduled after time τ except those tasks which were being executed at time τ .

Fig. 2. Reconstruction steps for failure/recovery at time τ

G	# Nodes	Graph Type	G	# Nodes	Graph Type	G	# Nodes	Graph Type
g1	15	BH_LD_N	g2	15	BH_LD_E	g3	50	BH_LD_N
g4	50	BH_LD_E	g5	100	BH_LD_N	g6	100	BH_LD_E
g7	15	BH_SD_N	g8	15	BH_SD_E	g9	50	BH_SD_N
g10	50	BH_SD_E	g11	100	BH_SD_N	g12	100	BH_SD_E
g13	15	TH_LD_N	g14	15	TH_LD_E	g15	50	TH_LD_N
g16	50	TH_LD_E	g17	100	TH_LD_N	g18	100	TH_LD_E
g19	15	TH_SD_N	g20	15	TH_SD_E	g21	50	TH_SD_N
g22	50	TH_SD_E	g23	100	TH_SD_N	g24	100	TH_SD_E

BH: Bottom Heavy graph LD: Long edges N: Node weights > edge weights
 TH: Top Heavy graph SD: Short edges E: Edge weights > node weights

Table 1. Table for test graphs

clearly show that in general the DSH-extension works much better than the ISH-extension in our model.

4 Fast schedule switching using pre-computed schedules

In this section, we present two strategies which *pre-compute* a set of schedules requiring different number of processors. Consider a situation where the current schedule is executing on a set of p processors, and a failure occurs. Instead of recomputing a new schedule with $p - 1$ processors, we now already have a precomputed schedule with $p - 1$ processors. What we need to determine at this

time is the mapping between the $p - 1$ clusters in the pre-computed schedule and the currently available $p - 1$ processors, that is, to determine which cluster will execute in which processor. This is clearly an instance of a weighted bipartite matching problem. However, strategies may differ in terms of computing the weights of the matchings. We present two simple strategies to compute the weight of the matching between cluster i in the pre-computed schedule with processor j at time τ .

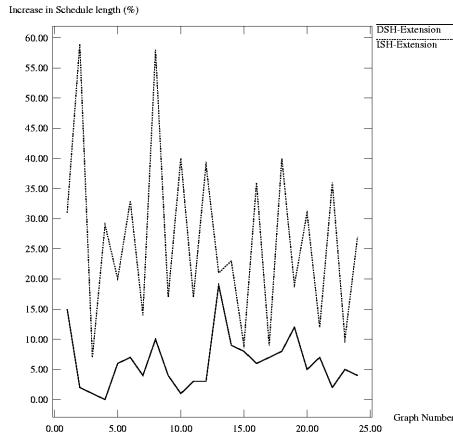
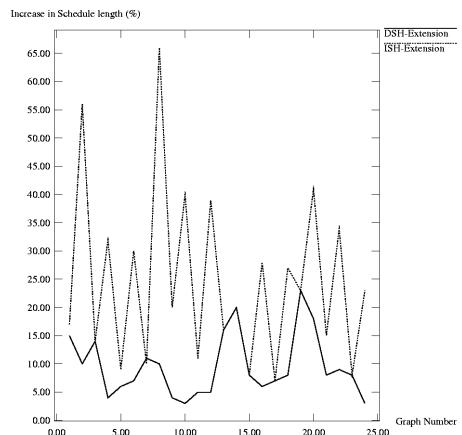
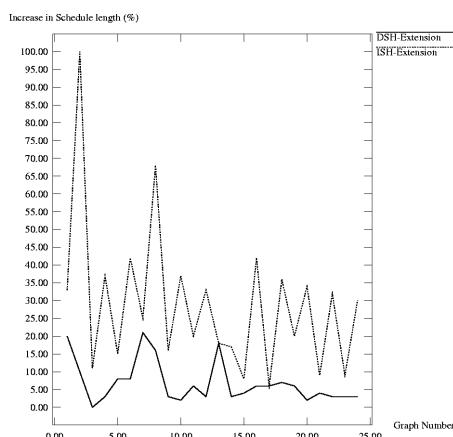
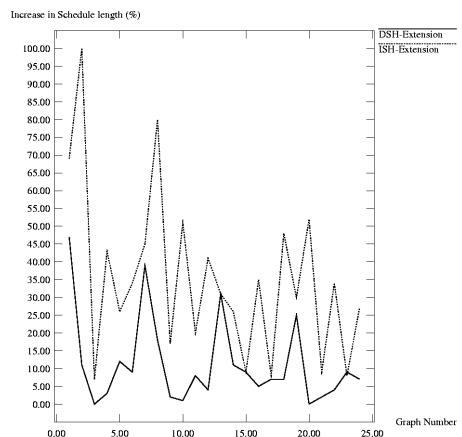
Completion based matching (CBM): Determine the set of tasks, S_τ , which are not *dropped*, but have completed or are currently executing in processor j at time τ . Compute the sum of the execution times of the tasks in the intersection of S_τ and the cluster i . This sum is taken as the weight of the matching between cluster i and processor j . We do not consider the *dropped* tasks because the data required from these are globally available. A maximum weight bipartite matching algorithm is used to determine the best mapping of clusters to processors.

Delay based matching (DBM): We try to schedule the remaining tasks in cluster i in the processor j as early as possible ignoring the timing requirements with respect to the tasks scheduled in other processors. As in CBM, we ignore the *dropped* tasks. While constructing this schedule, some of the tasks in cluster i may start late as compared to their start time in the original $p - 1$ processor schedule. We compute the maximum of these delays among the tasks in cluster i , and take that as the weight of the matching between cluster i and processor j . A minimum weight bipartite matching algorithm is used to determine the best mapping of clusters to processors.

The CBM and DBM algorithms were executed on the graphs of Table 1. As before the initial number of available processors for 15 node graphs was 4 and for the rest it was 10. Figure 8, Figure 9 and Figure 10 show the comparisons between DSH-extension, CBM and DBM in terms of percentage performance degradation with respect to the DSH schedules. Figure 8, Figure 9 and Figure 10 show the comparison when a failure occurs respectively at the 10%, 50% and 90% time of the initial DSH schedules. The graphs illustrate that the schedule lengths produced by CBM and DBM are comparable to re-scheduling using DSH, but we gain, since the re-scheduling overhead is smaller.

References

1. Das, D., Dasgupta, P., Das, P.P., A heuristic for the maximum processor requirement for scheduling layered task graphs with cloning. *Journal of Parallel and Distributed Computing*, **49** (1998), 169-181.
2. Kruatrachue, B., and Lewis, T., Grain size determination for parallel programs. *IEEE Software*, Jan 1998, 23-32.
3. Palis, M.A., Liou, J.C., and Wei, D.S.L., Task Clustering and Scheduling for Distributed Memory Parallel Architectures. *IEEE Transactions on Parallel and Distributed Systems*, **7**, No 1, January 1996, 46-54.
4. Papadimitriou, C. H., and Yannakakis, M., Towards an architecture independent analysis of parallel programs. *SIAM J. Computing*, **19** (2), 1990, 138-153.
5. Rewini, H.E., Lewis, T., and Ali, H. H., *Task Scheduling for Parallel and Distributed Systems*, Prentice-Hall, 1994.

**Fig. 3.** DSH vs ISH (Failure at 10%)**Fig. 5.** DSH vs ISH (Failure at 90%)**Fig. 4.** DSH vs ISH (Failure at 50%)**Fig. 6.** DSH vs ISH (Failure at 10% followed by Failure at 50%)

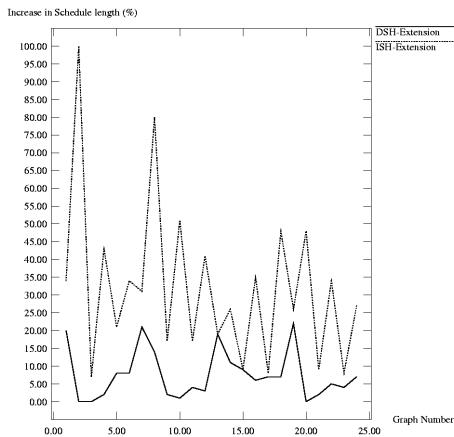


Fig. 7. DSH vs ISH (Failure at 10% followed by Failure at 50% and Recovery at 60%)

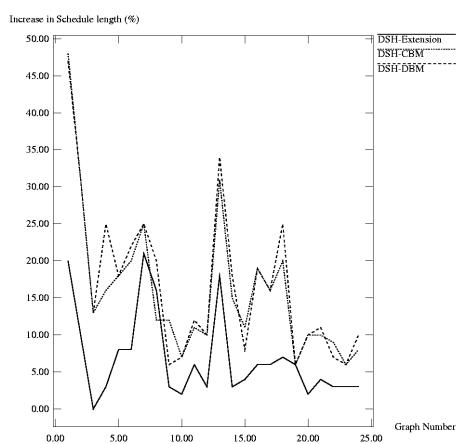


Fig. 9. DSH-Extension vs Matching (Failure at 50%)

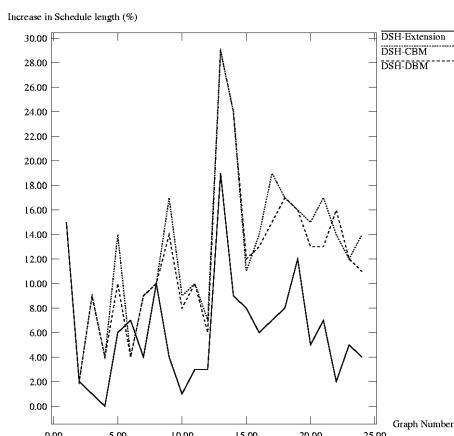


Fig. 8. DSH-Extension vs Matching (Failure at 10%)

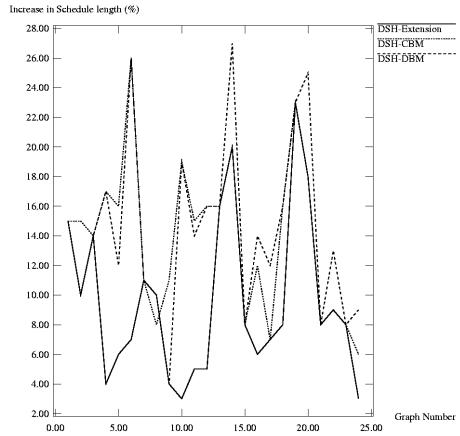


Fig. 10. DSH-extension vs Matching (Failure at 90%)

Scheduling Strategies for Controlling Resource Contention on Multiprocessor Systems

Shikharesh Majumdar

Department of Systems and Computer Engineering,
Carleton University,
Ottawa, CANADA K1S 5B6
email: majumdar@sce.carleton.ca

Abstract. Multiple processes may contend for shared resources such as variables stored in the shared memory of a multiprocessor system. Mechanisms required to preserve data consistency on such systems often lead to a decrease in system performance. This research focuses on scheduling policies that control shared resource contention and achieve high capacity and scalability in multiprocessor-based applications that include telephone switches and real-time databases. Based on analytic models three different scheduling approaches are analyzed.

1 Introduction

Contention for shared resources such as memory, database records, and various other types of computing resources is a well-known problem in the context of concurrent applications running on a multiprocessor. Locks and semaphores are used for serializing access to shared variables and preserving data consistency. For example, a transaction on the telephone switch locks a shared memory address before using it [2]. The locking protocol [8] is very similar to that on a database system. All the addresses locked by a transaction are released when the transaction commits. Whenever a transaction attempts to access a shared memory address which is already locked by a different transaction it rolls back and releases all the addresses that were locked by it. The transaction restarts from the beginning at a future point in time after the completion of the transaction that held the desired memory locations. Locks are used for controlling access to shared resources on general purpose computing systems as well. If a lock request fails because the resource is used by another process the requesting process is rolled back. For example, a database transaction needs to lock a record before using it. Many systems that include databases and the telephone switch discussed earlier roll back a process with a failed lock request for avoiding deadlocks. Various strategies exist for choosing a process for rollback (see [9] for example). Rollbacks can impede system capacity and need to be effectively controlled. Rollbacks due to shared memory contention on a multiprocessor-based call control module, for example, can limit the capacity of a telephone switch [7]. The techniques discussed in this paper are primarily concerned with

improving throughput or capacity of a telephone switch, a database, or other multiprocessor-based applications discussed in the previous paragraph.

The workload processed by the multiprocessor system varies from application to application. This research is based on the assumption that the workload consists of two broad classes of processes: *primary* and *secondary*. The primary workload component corresponds to the main functionality of the application whereas the secondary component corresponds to subsidiary activities. It is possible to extend the analytic model to more process classes but the the two-class system is appropriate for characterizing many systems and is apt for answering the high level questions analyzed in this paper. In a database system, for example, the primary class of processes correspond to database transactions whereas the secondary processes may correspond to backups and other maintenance related activities (such as checkpointing). On a telephone switch the primary class consists of processes that perform call processing and billing whereas the secondary process class performs support activities such as switch maintenance. A call processing process executes several transactions each of which corresponds to a particular call processing phase. In the following discussion we will be referring to computational units in the primary processes as transactions irrespective of whether they occur on a telephone switch, a database system, or some other multiprocessor-based application characterized by a set of primary and secondary processes.

This paper focuses mainly on two types of process contentions: secondary-primary and secondary-secondary. The scheduling methods discussed in the paper are most effective for controlling such interactions. Since primary processes receive a large proportion of the CPU power reducing interaction among such processes often require re-engineering of code. Extension of the scheduling techniques to control the remaining primary-primary contention are discussed in [6].

A considerable amount of work exists in the area of resource contention control (see [1] for example). Management of processing resources on multiprocessor systems has spanned two different domains: the allocation of processors to the threads in a given parallel computation [10] and the scheduling of processors to multiple parallel applications running simultaneously on the system (see [3] for a survey). None of these works has used process scheduling for reducing resource contention that this research focuses on. Based on an analytic model this paper is concerned with the understanding of the relative performances of three scheduling strategies that generate process schedules for reducing resource contention on a telephone switch and other miltiprocessor-based systems. The paper is organized as follows. The system model and the scheduling policies are described in the next section. Expressions for system capacity under different scheduling strategies are presented in Section 3. Numerical results obtained from the analytic model are discussed in Section 4. Section 5 presents our conclusions.

2 System Model and Scheduling Policies

The system model is characterized by a number of parameters that include system attributes as well as contention characteristics of the workload. An important parameter of the system model is N , the number of processors in the system. Given N , we are concerned with the effect of resource contention on the *useful capacity* of the system. The useful capacity E of the system is the average number of processor seconds per second spent in primary processing:

$$E = N - W_{oh} - W_{sp} - W_{rb} \quad (1)$$

where W_{oh} , W_{sp} , and W_{rb} are processor seconds per second spent on an average due to various operating system overheads, for running secondary software, and due to rollbacks respectively. A fair share scheduling approach is used. The secondary software is given a fixed share of the processing resources as defined by r which is the proportion of CPU time given to the secondary processes. Contention for shared resources is characterized by three contention parameters each of which is a conditional probability.

q_{xy} : the conditional probability that a process in class x contends for a resource with a process in class y when a class x and a class y process run simultaneously on the system.

Setting x and y to p (primary) and s (secondary) results in the three contentions probabilities q_{pp} , q_{sp} , and q_{ss} .

Scheduling Policies

The queue for ready processes that are run to completion on the system are maintained in shared memory. A mix of primary and secondary processes as determined by the ratio r is run. The following scheduling strategies are considered.

Serial: All secondary processes are run serially. That is, at a given point in time only one secondary process can run on the system. Primary processes are run in parallel.

Independent: All processes (primary and secondary) are run in parallel. Any ready process can run at the same time as any other process in the system.

Gang: A gang is defined to be a set of secondary processes. All the processors in the system are to be devoted to running members of the gang and no other process should run until the execution of the gang is complete. Upon completion of the gang primary processes are again run in parallel on the system. Policies similar to Gang scheduling described in this paper are considered in the context of achieving quick response times for scientific computations (see [4] for example).

3 Computation of System Capacity

The system capacity achieved with the various scheduling policies is discussed. Due to limitation of space only the expressions for capacity are presented. A detailed derivation of these expressions is provided in [6]. The computation of system capacity is based on the following assumptions that are appropriate for the high level research presented in this paper.

- All overheads are ignored and $W_{oh} = 0$.
- Whenever a collision occurs between a primary and a secondary process the primary process rolls back.

A more detailed discussion of these assumptions is provided in [6]. The expression for system capacity for each scheduling strategy is presented next.

The Serial scheduling Strategy:

$$E = N - Nr - Nr [(N-1)(N-2) \frac{q_{pp}}{4} + (N-1)\frac{q_{sp}}{2}] - [1 - Nr] N (N-1) \frac{q_{pp}}{4} \quad (2)$$

The Independent Scheduling Strategy:

$$E = N - Nr - \sum_{P=0}^N r^{(N-P)} (1-r)^P C(N, P) [C(P, 2) \frac{q_{pp}}{2} + (N-P)P \frac{q_{sp}}{2} + C(N-P, 2) \frac{q_{ss}}{2}] \quad (3)$$

where,

$C(J, K)$ = Number of ways in which K items can be chosen from J items (J Choose K)

The Gang Scheduling Strategy:

$$E = [1 - \frac{r}{1 - (q_{ss}(N-1))/4}] [N - N(N-1) \frac{q_{pp}}{4}] \quad (4)$$

4 Performance of the Scheduling Policies

N is varied from 1 to 10 processors in most experiments. Such values of N correspond to a number of commercial-off-the-shelf multiprocessor systems that provide an attractive price performance ratio. We have experimented with different combinations of workload parameters and for conservation of space one representative sample is presented in the paper. In order to gain insight into secondary-primary interaction which this paper focuses on q_{pp} is fixed at 0. Non-zero values of q_{pp} are discussed in [6]. The capacity E attained with Serial, Independent, and Gang for $q_{sp} = 0.5$ and $q_{ss} = 0.25$ are presented as a function of the number of processors N in Figure 1. Primary processes correspond to the main system functionality and give rise to the major proportion of system workload. We have varied the proportion of secondary work r from 0.05 to 0.2. When $r = .05$ the impact of secondary-primary interaction is small and all the policies

demonstrate comparable performance (see Figure 1(a)). However at larger values of r Gang demonstrates a superior performance in comparison to Independent and Serial (see Figure 1(b)). Note that Serial can not run at $N > 1/r$ because at higher values of N the rate of secondary work exceeds 1 CPU-sec/sec and cannot be run purely sequentially. For $r = 0.2$ and $N < 1/5$, Serial demonstrates the worst performance. Gang demonstrates the best performance followed by Independent. Note that no secondary-secondary contention occurs in case of Serial which runs all the secondary processes serially. Similarly no secondary-primary collision occurs in case of Gang because primary and secondary processes are run in a mutually exclusive manner. Since q_{ss} is small compared to q_{sp} , Gang demonstrates the best performance. Because with Independent the secondary processes can run in parallel the overall period of time for which primary and secondary execute simultaneously on the system is smaller than that in case of Serial. Although Independent is susceptible to both secondary-primary as well as secondary-secondary interactions the reduction in the overlap period of primary and secondary interaction seems to dominate performance and Independent performs better than Serial. Analyses of data points characterized by higher values of q_{sp} and q_{ss} are presented in Table 1. On systems characterized by high q_{sp} and low q_{ss} (Case 1) Gang is observed to exhibit the best performance. On systems with low q_{sp} and high q_{ss} (Case 2) Serial produces the highest capacity whereas Independent ranks the first when both q_{ss} and q_{sp} are high (Case 3).

4.1 Sensitivity to Changes in Workload Parameters

The resource contention characteristics of secondary as well as primary software can vary during system operation. Changes in workload or in the execution pattern of secondary software resulting from the occurrence of a fault may be responsible for such a change. A good scheduling strategy should be robust to such fluctuations in resource contention characteristics and variabilities in system capacity arising from such fluctuations should be minimal. The performance of the three strategies for different values of q_{sp} and q_{ss} are presented for $N = 5$ in Figure 2. A medium value of $r = 0.15$ is used. No secondary-secondary interaction is possible in Serial that is insensitive to changes in q_{ss} but is sensitive to changes in q_{sp} . Since secondary-primary interaction is not possible Gang demonstrates insularity from changes in q_{sp} but is sensitive to changes in q_{ss} . It is interesting to note that the capacity attained with Independent changes only marginally when q_{ss} is changed from 0 to 0.6 (see Figure 2(b)). Although q_{sp} has a stronger influence on the performance of Independent in comparison to q_{ss} , the sensitivity of Independent to changes in q_{sp} is smaller than that exhibited by the Serial scheduling strategy (see Figure 2(a)).

5 Conclusions

Accesses to shared resources by multiple processes need to be serialized for maintaining data consistency on a multiprocessor system. The lock-rollback mechanism used in systems for achieving such serialization in data access can lead to

serious limitation in system performance on a multiprocessor-based system. Both reengineering of existing code and scheduling of processes for avoiding resource contentions are viable methods for achieving scalable power. An investigation of process scheduling to reduce secondary-primary and secondary-to-secondary interactions is described in this paper. A comparison of the three scheduling approaches and insights gained into system behavior are presented in this section.

Both secondary-primary as well as secondary-secondary interactions are possible with the Independent scheduling strategy. The impact of q_{sp} is observed to be stronger on the performance of Independent in comparison to q_{ss} . No secondary-secondary interaction is possible in case of the Serial scheduling strategy. The performance of Serial is observed to degrade, however, with an increase in q_{sp} (see Figure 2(a)). A nice feature of the Gang strategy is its insularity from the processes outside the gang. The performance of Gang is however sensitive to q_{ss} that results from the interaction among processes within the gang. A hybrid policy that combines the insensitivity properties of Serial and Gang is described in [6].

Overheads associated with scheduling were ignored in this preliminary analysis. The Independent strategy is likely to give rise to the least overhead on a symmetric multiprocessor in which each processor runs independently. Implementing the Serial and the Gang strategies require some kind of synchronization among the processors and results in additional overheads. A trace-driven simulation that captures the synchronization overheads associated with the implementation of Gang and Serial is underway [5].

Acknowledgements I would like to thank Nortel Networks for providing me with the opportunity for conducting this research. Thanks are due to Brian Carroll, Bruce Beninger, Nicole Klappholz, Tadeusz Drwiega, Sunderam Chintamani, Gordon Fitzpatrick, Brian Geddes, and Imran Ahmad for their comments and help in preparation of the paper.

References

1. Dandamudi, D., Eager, D.L., "Hot-Spot Contention in Binary Hypercube Networks", *IEEE Trans. on Computers*, Vol. 41, No. 2, February 1992.
2. Drwiega, T., "Shared Memory Contention and its Impact on Multi-Processor Call Control Throughput", *Proc. 15th International Teletraffic Congress*, June 1997, pp. 22-27.
3. Feitelson, D., "A Survey of Scheduling in Multiprogrammed Parallel Systems", Research Report RC19790 (87657), IBM T.J. Watson Research Center, New York, 1994.
4. Feitelson, D., "Packing Schemes for Gang Scheduling", in *Job Scheduling Strategies for Parallel Processing*, Springer Verlag, Lecture Notes in Computer Science, 1996.
5. Liu, M. "—" M.Eng. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, CANADA, K1S 5B6, January 2000 (expected).
6. Majumdar S., "Controlling Resource Contention on Multiprocessors: a Scheduling-Based Approach", Technical Report, Department of Systems and Computer Engineering, Carleton University, Ottawa, CANADA, K1S 5B6, September 1999.
7. Majumdar, S., Streibel, D., Beninger, B., Carroll, B., Verma, N., Liu, M., "Controlling Memory Contention on a Scalable Multiprocessor-Based Telephone Switch", *Proc. ACM SIGMETRICS '99 Conference*, (Extended Abstract), Atlanta, May 1999, pp. 228-229.
8. Newell, T.E., Baker, B., "Shared memory Control Algorithm For Mutual Exclusion and Rollback", Nortel Networks Patent (pending), 1996.
9. Singh, M. Shivaratri, N.G., *Advanced Concepts in Operating Systems*, McGraw-Hill, 1994.
10. Woodside C.M. and G.G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 164-174, Feb. 1993.

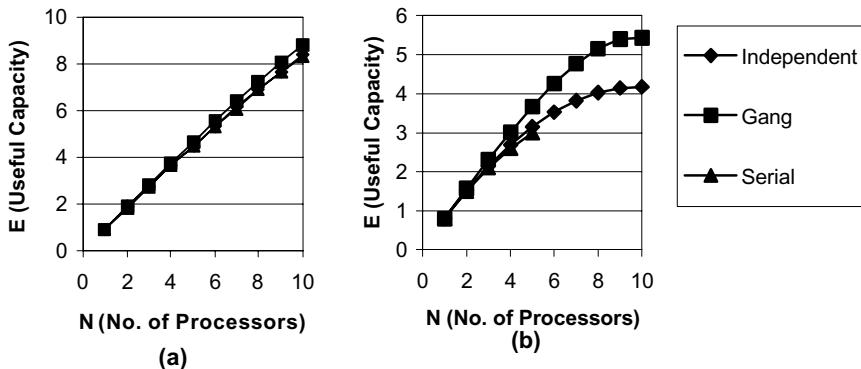


Figure 1. Performance Comparison ($q_{sp}=0.5$, $q_{ss}=0.25$) (a) $r = 0.05$ (b) $r = 0.2$.

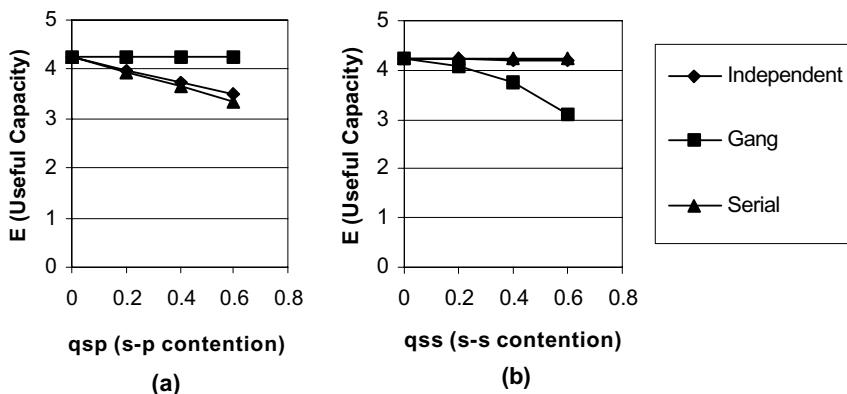


Figure 2. The Effect of Contention ($N = 5$, $r = 0.15$) (a) $q_{ss}=0$ (b) $q_{sp}=0$.

Table 1. Systems with High Contention.

Case	E (Independent)	E (Serial)	E (Gang)
Case 1: $q_{ss}=0$, $q_{sp}=0.6$	3.4	3.35	4.25
Case 2: $q_{ss}=0.6$, $q_{sp}=0$	4.182	4.25	3.125
Case 3: $q_{ss}=0.6$, $q_{sp}=0.6$	3.428	3.35	3.125

Deadline Assignment in Multiprocessor-Based Fault-Tolerant Systems

Sharath Kumar Kodase¹, N.V. Satyanarayana¹, R. Mall², and A. Pal¹

¹ Indian Institute of Technology, Kharagpur, 721 302, India

² Curtin University of Technology, GPO Box U1987, Perth WA 6845, Australia

e-mail: rajib@cs.curtin.edu.au

Abstract. Real-time tasks often consist of subtasks with complex interdependencies on which an end-to-end deadline applies. In such cases, in order to achieve fault-tolerance through scheduling, it becomes necessary to assign deadlines to the subtasks and schedule them on different processors. In this paper we examine the effectiveness of different strategies for deriving subtask deadlines from global deadlines through extensive simulation studies.

1 Introduction

Multiprocessor systems are the popular choices for implementing real-time systems mainly because of their speed, reliability, and their decreasing costs. Many fault-tolerant scheduling algorithms have been proposed in the literature for both static scheduling and dynamic scheduling [1-3],[6,7] in multiprocessor systems. A popular methodology has been the use of *primary-backup* model to provide fault-tolerance [1]. In this model, every task that is scheduled in the system has a backup version which runs on a different processor. The backup version is activated only when the primary version fails. A task succeeds when either the primary or the backup version completes successfully. A common assumption in these algorithms has been that tasks are atomic entities. Real-time systems are becoming larger and more complex as computers advance in speed and sophistication. Many of these systems have fault-tolerance requirements. As tasks become more complex and bigger in size it becomes necessary to subdivide them into a set of subtasks with an end-to-end deadline [5]. These subtasks can have complex interdependencies [4]. Most designers handle this situation by first allocating the overall deadline among the subtasks *ad hoc* and then iterating several times through the design procedure. However, this approach is clearly error-prone, time-consuming, and costly. In this context, we examine the effectiveness of different deadline allocation schemes for primary-backup fault-tolerant scheduling algorithms.

There are relatively few studies on the problem of deadline allocation to the subtasks of a real-time system. In a recent important work [4], Kao and Garcia-Molina studied the problem of deadline allocation to subtasks in a soft real-time environment. However, they do not consider the effect of different deadline allocation schemes on the fault-tolerant scheduling requirements which is the main focus of our present work.

In this paper, we consider this problem of allocating deadlines to the subtasks and study different strategies for allocating deadlines to the subtasks and their impact on the overall schedulability of the system. We study in particular fault-tolerant dynamic scheduling in a shared memory multiprocessor environment.

2 System and Task Model

We consider a shared memory multiprocessor system consisting on N identical processors and a global scheduler. Real-time tasks are considered to be directed acyclic graphs. We also assume that there is a single starting subtask (start node) and a single terminating subtask (end node) in the task. This can always be obtained with the use of dummy subtasks at the beginning and the end of the task. Tasks arrive at the global scheduler at a fixed arrival rate according to Poisson distribution. The global scheduler tries to schedule the task in a fault-tolerant manner among the processors. If a schedule is found by which the newly arrived task can be accommodated without affecting the guarantees to the previously scheduled set of tasks, then the task is accepted. Otherwise, it is rejected. The use of a global scheduler makes it a single point of failure in the system. This can be avoided by having a scheduler as a standby such that the standby can take over the scheduling when the global scheduler fails. We assume that there can at most be one fault at any point of time and a second fault does not occur in the system until the system has recovered from the first one. This assumption makes it sufficient for only two versions of a subtask to be scheduled on different processors in order to provide fault-tolerance. Also the fault rate is very low as compared to the task arrival rate. Therefore a subtask may encounter at most one failure during its execution. So, if the primary version of the subtask fails, backup version always succeeds. We also assume that the system takes some fixed time to recover from a fault. Till that time we try and provide fault-tolerance using the remaining set of processors.

3 Scheduling Algorithm

The algorithm we employ for fault-tolerant scheduling is a variation of the algorithm that was presented in [3] with fault-tolerant enhancements. In this algorithm a subtask in a task graph becomes active only if all its immediate predecessor subtasks have finished execution. This is required by the precedence constraints. So, at a given time, only a certain set of subtasks in a task will be active. The order in which these subtasks are considered for scheduling will determine if we will be successfully able to schedule the subtasks or not. In our approach we have ordered the tasks according to their latest start times (LST). If there are ties, we extract the subtasks one by one from the set of ready subtasks according to the LST/MISF (Latest Start Time/ Maximum Immediate Successor First) heuristics. We then schedule the primary first by finding the earliest slot where the primary can be scheduled. This is done by checking if any processor has a time slot of length $Ex(T_j)$ free between times $Ready(T_j)$ and $Deadline(T_j)$, where $Ex(T_j)$ is the worst case execution time of the sub task T_j . $Ready(T_j)$ is the ready time, and $Deadline(T_j)$ is the deadline of the subtask T_j . The ready time of a subtask is the time when it becomes active for execution. This is the time when all its predecessors have finished their executions (either their primary or backup). This implies that the ready time of a subtask will be the maximum of the scheduled end times of the backup slots of its predecessor subtasks. If no such slot is

found then the task is rejected. If many such slots are found then we employ first fit algorithm to find the processor on which to schedule.

In order to schedule the backups we employ two methods. In the first method, if the primary has been scheduled in slot $< T_1, T_2 >$, then we try and schedule the backup as soon as possible between times T_1 and $\text{Deadline}(T_j)$ on a different processor. If there are no slots between this period where the backup can be scheduled, then the task is rejected. If there are more than one slots, we use the first fit algorithm to choose the processor on which to schedule the backup. We call this approach as backup as soon as possible (BASP). Another variation of this scheme will be to schedule the backup as late as possible within the time interval T_1 and $\text{Deadline}(T_j)$. This approach is called backup as late as possible (BALP). After successfully scheduling the primary and the backup versions, we now add to the set of ready tasks all the subtasks that become active now. Then the entire process is repeated till all the subtasks have been scheduled.

4 Deadline Allocation for Subtasks

One of the simplest method of deadline allocation is to assign to each subtasks deadlines that is same as their latest finish time (LFT). This is a very natural method of assigning deadlines to the subtasks. We call this effective deadline policy (ED). But the main disadvantage of this policy is that all the slack available to the task will be given to initial subtasks. This leaves the remaining subtasks with no slack. Another method of assigning the deadlines is to distribute the slack to each subtasks according to their execution times (EQF). So a larger subtask will have a larger slack. This policy does not bias the initial tasks as ED policy does. We assign the slacks to the individual subtasks starting from the end node and proceeding in a reverse topological order backwards up to the start node.

We have also tried some very interesting deadline allocation strategies like biasing the smaller subtasks in favour of the larger subtasks (STF). In this strategy we gave smaller subtasks slack equal to their execution times first. The slack was distributed to the subtasks starting from the smallest subtask and proceeding toward larger subtasks. For each subtasks a slack equal to its execution time was given, if the required amount of slack was available or else no slack was given.

5 Performance Studies

We have carried out extensive performance studies of the different deadline assignment strategies we discussed in the earlier section and the fault-tolerant scheduling algorithms. In our experimentation, we use Acceptance Ratio (AR) defined as the ratio of tasks scheduled to the total number of tasks that have arrived as the measure of performance of the scheduling algorithms. A study of slack utilization can be made with the help of the graph between Acceptance Ratio vs Slack (Fig. 1). This graph was obtained by fixing the number of processors at 8 and task arrival rate at 0.02 (corresponding to a load of 0.6). It can be observed from this graph that the BALP algorithms make more efficient use of the slack. As the slack increases the AR of BALP algorithms increases to 1.0 irrespective of the deadline assignment strategy used. Since the BASP algorithms try and schedule the backup as soon as possible, the slack of the task is left unutilized and resource reclaiming is much less than BALP.

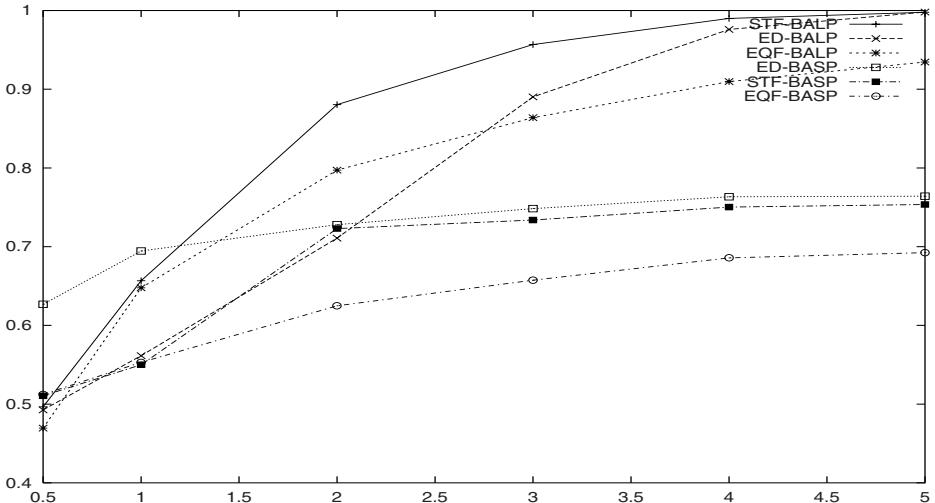


Fig. 1. Acceptance Ratio vs Slack

Consequently the AR of BASP algorithms is low when compared to BALP. But when the slack of the tasks is low, ED-BASP performs better than all the other algorithms. This is because the subtask deadline assigned by ED algorithms is always later than that assigned by the other algorithms. Hence the scheduler has relatively more time to schedule the subtasks.

From the graph of AR versus the subtask arrival ratio (Fig. 2), it can be observed that BALP algorithms perform better than BASP algorithms for all deadline allocation policies. Among the various deadline allocation policies that we have studied, STF policy when used with BALP was found to be the best. Since the STF policies favour smaller subtasks, more subtasks will have their backups scheduled passively. This leads to more resource reclaiming and hence, a better AR. Since all our algorithms are based on utilizing slack available to the task efficiently, when the slack is fixed, (as in the case of the AR vs arrival rate and AR vs number of processors) it can be observed that there is only a marginal difference between ED and EQF deadline allocation policies.

6 Conclusion

Most of the reported fault-tolerant scheduling algorithms make a tacit assumption that all tasks are atomic in nature. They assume that tasks are indivisible and are characterized by an arrival time, a deadline, and a computation time. In reality, the system designer is often confronted with real-time tasks which are much bigger and for better management of temporal properties, it has to be broken up into a set of events which are detectable.

In this paper, we have examined the scope of various deadline allocation policies by extending them to fault-tolerant scheduling for real-time systems. We have carried out extensive simulation studies of two fault-tolerant algorithms (BASP, BALP)

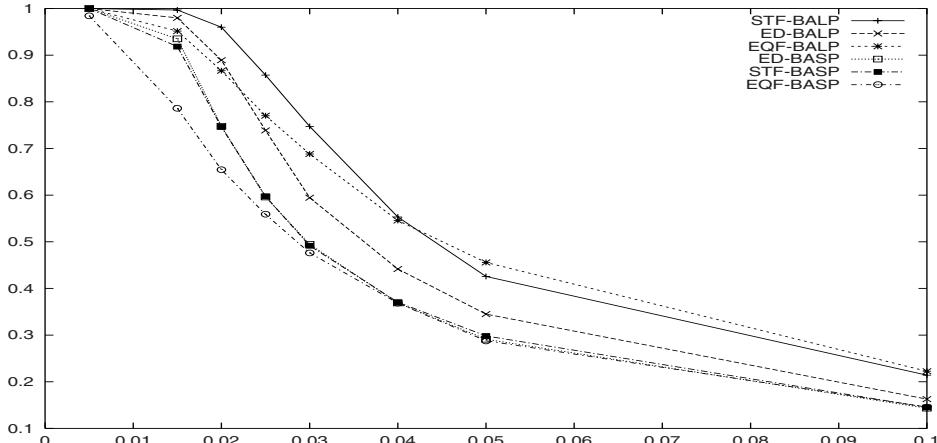


Fig. 2. Acceptance Ratio vs Task Arrival Rate

and three deadline allocation policies (ED,EQF,STF). Our simulation results reveal that deadline allocation policy can be important in effectively utilizing the slack and policies like STF improve the overall schedulability of the system by improving upon resource reclaiming. Algorithms that schedule the backups as early as possible (BASP) perform well with scheduling policy like ED for tasks with low slack (of order of less than 1). But for tasks with slack of greater than 1, we observe that a better scheduling policy is BALP along with STF and ED.

References

1. S.Ghosh, R.Melham, D.Mosse, Fault-Tolerant Scheduling on a Hard Real-Time Multiprocessor System, Proc. 8th International Parallel Processing Symposium, pp 775-782, 1994.
2. D. Mosse, R. Melham, S. Ghosh, Analysis of a Fault-tolerant Multiprocessor Scheduling Algorithm, Proc. 24th International Symposium on Fault-tolerant Computing, pp 16-25, 1994.
3. T.Tsuchiya, Y.Kakuda, T.Kikuno, Fault-Tolerant Scheduling algorithm for distributed Real-Time Systems, Proc. Proc. 3rd Workshop on Parallel and Distributed Real-Time Systems, pp 99-103, 1995
4. K. Ramamritham, Allocation and Scheduling of Precedence-Related Periodic tasks, IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 4, April 1995.
5. B. Kao, H. Garcia-Molina, Deadline Assignment in a Distributed Soft Real-Time System, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 12, December 1997.
6. C.M. Krishna, KG Shin, On Scheduling Tasks with Quick Recovery from Failure, IEEE Transactions on Computers, vol.35 no. 5, pp448-455, May 1986.
7. A.L. Leistman and R.H. Campbell, A fault-tolerant scheduling problem, Trans. Software Engg., SE-12(11),pp. 1089-1095, November 1988.

Affinity-Based Self Scheduling for Software Shared Memory Systems¹

Weisong Shi and Zhimin Tang
Institute of Computing Technology
Chinese Academy of Sciences
E-mail:{wsshi,tang}@water.chpc.ict.ac.cn

Abstract. In this paper, we propose an affinity-based self scheduling scheme (ABS) for software shared memory system. In this scheme, the static affinity between processor and initial data distribution is considered when scheduling, and the synchronization overhead is reduced greatly when load imbalance occurs. Compared with previous schemes, ABS performs better in metacomputing environment.

1 Introduction

Loop scheduling has been extensively studied in past years, and many schemes were proposed, such as Self Scheduling(SS) [7], Block Self Scheduling (BSS), Guided Self Scheduling (GSS)[5], Factoring Scheduling (FS) [2], Trapezoid Self Scheduling (TSS) [8], Affinity Scheduling (AFS) [4], Safe Self Scheduling (SSS) [3], Adaptive Affinity Scheduling (AAFS) [9], etc.. However, all these previous work focused on either shared memory multiprocessors or dedicated hardware distributed shared memory systems, i.e., they assumed that the available computing power of each processor was equal, and neglected the practical computing power of underlying hardware. Therefore, results are not applicable under metacomputing environment. Take matrix multiplication as an example, the static scheduling scheme is demonstrated as the best among all the schemes by previous research. However, we will show that static scheduling is the worst scheme in metacomputing environment.

In this paper, we present and evaluate a new affinity-based (ABS) dynamic self scheduling algorithm for home-based software DSM system. The unique characteristic of ABS scheme, the static affinity between processor and initial data distribution, is considered when scheduling. The evaluating results show that ABS performs better than all previous scheduling schemes in meta-computing environment.

2 Motivation

All those dynamic scheduling algorithms discussed above fall into two distinct classes: *central queue based* and *distributed queue based*. In central queue based algorithms, such as SS, BSS, GSS, FS, TSS, SSS, iterations of a parallel loop are all stored in a shared central queue and each processor exclusively grabs some iterations from the central queue to execution. The major advantage of using a central queue is the possibility of optimally balancing the load. However, the processor affinity is neglected completely in this kind of schemes, which performs worse in distributed memory system[9]. Although distributed queue based

¹ The work of this paper is supported by the National Natural Science Foundation of China under Grant No. 69896250 and Microsoft Research China Fellowship.

schemes take the processor affinity into consideration, the extra synchronization overhead associated with AFS and AAFS when load imbalance occurs are neglected in previous work. In distributed systems, this operation requires at least $2P + 2$ messages and $P + 1$ synchronization operations, where P is the number of processors. This operation requires much more time and will affect the final execution time greatly. Moreover, the processor affinity exploited in AFS and AAFS is limited to iterative scientific applications only.

3 Design and Implementation of ABS

In home-based software DSM system, the data must be predistributed, so that the affinity between each iteration and its corresponding data can be determined statically. Therefore, this initial affinity information can be used in the following scheduling scheme.

The basic idea of affinity-based self scheduling (ABS) is to distribute the loop iterations to the processor that already has corresponding data while minimizing the loop allocation overhead (i.e., synchronization overhead to access exclusive shared variables). In traditional central queue based scheduling algorithms, the initial data distribution is separated from the scheduling algorithm so that the affinity between the initial data distribution and the following computing can not be exploited, since the order of each processor visiting the central queue is not deterministic.

ABS includes three steps: *static allocation*, *local dynamic scheduling*, and *remote dynamic scheduling*. We implement a globally shared central queue, which is partitioned into p segments evenly, and the i th segment is assigned to processor i statically according to the initial data distribution. At local dynamic scheduling phase, BSS or GSS scheduling algorithms can be employed². When load imbalance occurs, the idle processor obtains the task from the most heavily loaded processor. It seems that there is no much difference between ABS and AFS at the later scheduling phases. However, the use of central shared queue, the lazy memory consistency model adopted by software DSM system, and the large granularity inherent in the software DSM system contributes greatly to our ABS scheduling algorithm. The large grain (generally one page is larger or equal to 4K bytes) causes the whole shared queue to be allocated in one page. Therefore there is no difference between local scheduling and remote scheduling, both of them require only one synchronization operation and one message. Table 1 compares the number of messages required and synchronization operations of ABS, AFS and AAFS. From the table, we conclude that when load imbalance occurs, more messages and synchronizations are required in AFS and AAFS to gather other processors' load information. We can also find that the synchronization overhead associated with the loop allocation of AAFS is worse than that of AFS, which will be demonstrated by the performance evaluation results in the next section.

² Due to space limitation, we present the results of GSS here only, more evaluation about different schemes adopted in the local scheduling phase of ABS are presented in [6]

Table 1. # of messages and synchronization operations associated with loop allocation

Scheme	Local scheduling		Remote scheduling	
	# of message	sync.	# of message	sync.
AFS	0	1	$2p+2$	$p+1$
AAFS	$2p$	$p+1$	$2p+2$	$p+1$
ABS	0 or 2	1	4	2

The main difference between ABS algorithm and traditional central queue based scheduling schemes lies in the organization and management of the central queue. In our scheme, the queue is partitioned among these processors, and each processor first accesses it's own part. In traditional central queue based scheduling algorithms, the initial data distribution is separated from the scheduling algorithm so that the affinity between the initial data distribution and the following computing can not be exploited at all.

4 Experiment Results and Analysis

4.1 Experiment Platform

The evaluation is done in the Dawning-1000A parallel machine developed by the National Center of Intelligent Computing Systems. The machine has eight nodes each with a 200MHz PowerPC 604 processor (the peak performance of each processor is 0.4Gflops) and 256MB memory. These nodes are connected through 100Mbps switched Ethernet. The software DSM system used in our evaluation is JIAJIA system[1], which is a home-based software DSM system. All the schedulings are implemented in JIAJIA. More information about JIAJIA can be found at <http://www.ict.ac.cn/chpc/dsm>.

We choose five applications which cover the range of all applications used in the related literature. The selected applications includes SOR, Jacobi Iteration (JI), Transitive Closure (TC), Matrix Multiplication (MM), and Adjoint Convolution (AC). Details of these programs can be found in [9]. Table 2 illustrates the basic characteristics of these 5 applications.

Table 2. Characteristics of the applications

Application	SOR	JI	TC	MM	AC
Size	4096, 10 iter.	5000, 10 iter.	2048×2048	2048×2048	256^2

4.2 Metacomputing Environment

For comparison, we add several artificial loads to some processors when testing. Figure 1 illustrates the execution time of different schemes under metacomputing environment. The loop allocation overhead is listed as the number of synchronization operations in Table 3. The number of remote getpages, which reflects the effect of remote data communication, is shown in Table 4. Here, the *Syn.* overhead includes two parts. One is the synchronization overhead associated

Table 3. The number of synchronization operations of different scheduling algorithms in metacomputing environment

Apps.	Static	SS	BSS	GSS	FS	TSS	SSS	AFS	AAFS	ABS
SOR	0	41040	410	610	310	270	270	7003	14414	970
JI	0	50080	420	620	315	270	288	7056	7251	3130
TC	0	205600	2500	5500	3154	2700	2761	56924	123529	5788
MM	0	2056	41	55	31	27	26	1040	1565	227
AC	0	65544	4105	81	31	27	41	1411	2595	465

with loop allocation operation, the other is the waiting time at synchronization point because of load imbalance.

As analyzed in [6], load imbalance is less important than locality in software DSMs. Therefore, for three kernels with fine computation granularity, SOR, JI, and TC, Static scheduling scheme remains well since the processor affinity is remained in Static scheme with respect to some dynamic schemes, such as SS, GSS etc. However, for coarse grain applications and applications which have limited locality, the performance of Static scheme becomes unacceptable because of the existence of load imbalance, such as MM and TC. Figure 1 demonstrates our analyses.

Though SS promises the perfect load balance among all the nodes, SS remains the worst scheme for all applications except MM because of the large loop allocation overhead, as shown in Table 3. Furthermore, Table 4 shows the inherent drawback of traditional central queue based scheduling algorithm , i.e., potential to violate processor affinity, results in large count of remote getpage operations. SS performs better than Static for MM because the synchronization waiting time dominates the whole synchronization overhead here. The corresponding *Syn* value reduces from 157.13 in Static to 8.07 in SS in MM kernel.

Table 4. The number of getpages of different scheduling algorithms in metacomputing environment

Apps.	Static	SS	BSS	GSS	FS	TSS	SSS	AFS	AAFS	ABS
SOR	8456	260838	67418	82892	77177	75212	81928	12091	10478	6215
JI	13820	182889	75719	70413	77531	75621	57852	9045	7251	4058
TC	2392	335924	54418	55878	82701	54153	22773	41511	49191	19129
MM	13547	33910	23843	26391	22965	21846	21935	17998	18198	17550
AC	100	58032	838	818	205	209	175	804	775	465

BSS, GSS, FS, TSS, and SSS make some progress compared with SS, especially for MM kernel. The performance of these 5 scheduling schemes is acceptable. However, as discussed in the Section 2, due to the large extra overhead resulting from loop allocation and corresponding potential of violating processor affinity associated with BSS, GSS, FS, TSS, and SSS scheduling schemes, as listed in Table 3 and Table 4, the performance of these 5 schemes remains unacceptable with respect to Static scheduling scheme in metacomputing environment for three fine grain kernels, which is illustrated in Figure 1. Among these

five scheduling schemes, SSS is the worst due to the large chunk size allocated in the first phase, which results in large amount of waiting time at synchronization points. For example, the *Syn* value of SSS is (*SOR*, 122.50), (*JI*, 138.71), while the corresponding values in FS are (*SOR*, 32.84), (*JI*, 65.00)³. Therefore, the conclusions proposed in [3] are no longer available. However, from the Figure 1, we find the FS is prior to GSS in metacomputing environment as claimed in [2].

For all five applications but TC, the performance of AFS is improved significantly compared with the former 6 dynamic scheduling schemes, as shown in Figure 1. Though the number of synchronization operations of AFS increases about one order of magnitude, as shown in Table 3, the number of remote get-page reduces about one order of magnitude, as listed in Table 4, which amortizes the effects of loop allocation overhead. Surprisingly, the number of remote get-page operations of AFS in TC leads to the contrary conclusion, i.e., it does not reduce at all with respect to former scheduling schemes. The main culprit here is the relaxed memory consistency model used in state-of-the-art software DSM system, where all the data associated with one synchronization object will be invalidated when requesting this synchronization object. Therefore, the large number of synchronization operations of AFS will lead to large amount of invalidation, which in turn leads to large number of remote fetchs.

As described before, AAFS increases the number of synchronization overhead in local scheduling phase in order to collect the states of other nodes. Therefore, the corresponding synchronization overhead becomes larger than that of AFS. For example, in TC, the number of synchronizations increases from 56924 of AFS to 123529 of AAfs, the corresponding synchronization time increases from 212.96 seconds in AFS to 541.22 seconds in AAfs. Other results are presented in Figure 1, Table 3 and Table 4. Our results give the contrary conclusion to that presented in [9] where AAfs was better than AFS for all five applications in their two hardware platforms.

5 Conclusions

In this paper, we have proposed a new affinity-based self scheduling cheme for software shared memory system. Based on the evaluation and analyses, we have observed the following results. SS is an inappropriate scheduling scheme for software DSM systems. BSS, GSS, FS, SSS and TSS perform better than Static and SS schemes. AFS, AAfs, and ABS achieve better performance because of the use of processor affinity. However, the advantage of AAfs with respect to AFS does not exist at all. ABS achieves the best performance among all scheduling schemes in metacomputing environment because of the reduction of synchronization overhead and the great improvement of waiting time resulting from load imbalance.

References

- W. Hu, W. Shi, and Z. Tang. Jiajia: An svm system based on a new cache coherence protocol. In *Proc. of the High Performance Computing and Networking Europe 1999*

³ Since the algorithm of FS and SSS is very similar except the value of chunk size in allocation phase, we compare it with FS here.

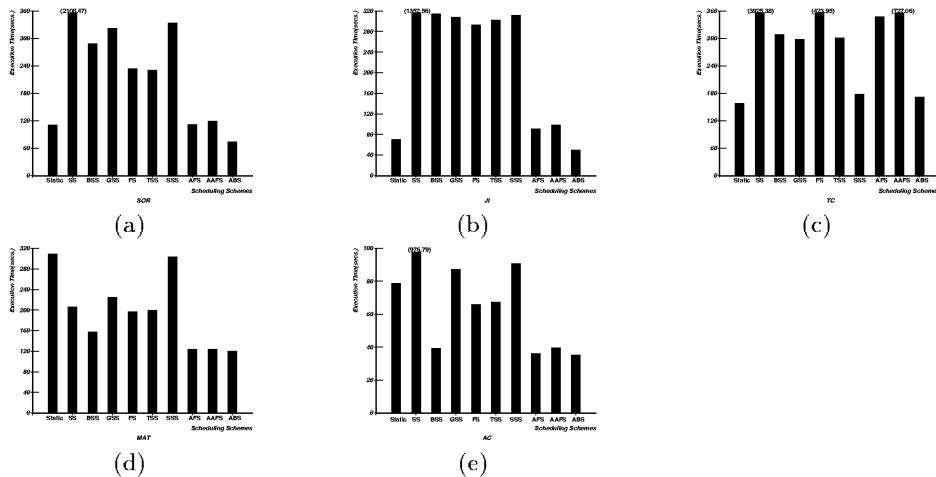


Fig. 1. Comparison of execution time of different scheduling schemes in metacomputing environment:(a)SOR, (b) JI, (c) TC, (d) MM, and (e) AC

- (HPCN'99), pages 463–472, April 1999.
2. S. E. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A practical and robust method for scheduling parallel loops. *Communication of ACM*, 35(8):90–101, August 1992.
 3. J. Liu, V. A. Saletore, and T. G. Lewis. Safe self-scheduling: A parallel loop scheduling scheme for shared memory multiprocessors. *Int'l Journal of Parallel Programming*, 22(6):589–616, June 1994.
 4. E. Markatos and T. Le Blanc. Using processor affinity in loop scheduling on shared memory multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 5(4):379–400, April 1994.
 5. C. Polychronopoulos and D. Kuck. Guided self-scheduling: A practical self-scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, 36(12):1425–1439, December 1987.
 6. W. Shi, Z. Tang, and W. Hu. A more practical loop scheduling for home-based software dsms. In *Proc. of the ACM-SIGARCH Workshop on Scheduling Algorithms for Parallel and Distributed Computing—From Theory to Practice*, June 1999.
 7. P. Tang and P.-C. Yew. Processor self-scheduling for multiple nested parallel loops. In *Proc. of the 1985 Int'l Conf. on Parallel Processing (ICPP'86)*, pages 528–535, August 1986.
 8. T. H. Tzen and L. M. Ni. Trapezoid self-scheduling:a practical scheduling scheme for parallel compilers. *IEEE Trans. on Parallel and Distributed Systems*, 4(1):87–98, January 1993.
 9. Y. Yan, C. Jin, and X. Zhang. Adaptively scheduling parallel loops in distributed shared memorysystems. *IEEE Trans. on Parallel and Distributed Systems*, 8(1):70–81, January 1997.

Efficient Algorithms for Delay Bounded Multicast Tree Generation for Multimedia Applications

Nishit Narang, Girish Kumar, and C.P. Ravikumar

Indian Institute of Technology, New Delhi 110016, India,
nisnarang@hss.hns.com, rkumar@ee.iitd.ernet.in

Abstract. Given a network topology and costs associated with the links, the problem of generating a minimum cost multicast tree can be modelled as a Steiner tree problem. However, many real time applications such as video-conferencing require that data be sent within prespecified *delay limits* in order to avoid problems such as anachronism and lack of synchronization. This paper deals with the delay-bounded cost-optimal multicast tree (DBCMT) generation problem. A closely related problem is to find a delay-bounded cost-optimal *path* (DBCP) between a specified source and destination node. Such a path can be used as a starting point to solve the DBCMT. We present here two heuristics for building delay constrained multicast trees which have near optimal cost. A comparison of our heuristics with other proposed heuristics is also presented.

1 Introduction

We consider the problem of obtaining a multicast tree of optimal cost in which the delay along the path connecting any pair of nodes is bounded from above. This problem is closely related to the Steiner Tree problem in graphs and must be solved using heuristics due to its NP-complete nature [2, 3]. Due to lack of space, we omit a survey of related work and refer the reader to [4]. We use the "dboc" algorithm [4] to find the delay bounded minimum cost path between two nodes. We propose the following two heuristics for DBCMT. Given a graph G , a tree T , and delay tolerance Δ , let us first define the *residual delay* at a particular node in the tree. The *residual delay* at a node n in a given tree T is the difference between the delay tolerance Δ and the maximum of the delays along T from n to any other node in T . In Figure 1, the residual delay at node c is 4.

1.1 DCBH : Dynamic Center Based Heuristic

This heuristic attempts to find a central node of the given multicast group and then joins all group members to the center one by one. The procedure determines minimum delays for all pairs of multicast nodes and finds the pair of nodes that has the maximum value. The center of the minimum delay path between these two nodes is identified as the initial center C . Each multicast node A is joined to the center as follows. The cost and delays of minimum delay path between A and C , and delay constrained minimum cost path (again between A and C) are found. A cost factor and a delay factor are computed. If the cost factor outweighs the delay factor, then we find a minimum cost path bounded

by delay equal to the average of the delays of the two paths. If cost factor is not high, we use the minimum delay path. This approach of not consuming the entire residual delay has two advantages. First, it substantially increases the extensibility of the partial tree which means that more number of multicast nodes can now join in. Secondly, it makes the center of the tree more mobile. The new node must now be added to the partial tree using the path just found.

1.2 BERD : Best Effort Residual Delay

Here, we store the residual delay at each node in the tree. To join an existing multicast group, a node finds a minimum cost *residual delay bounded* path to each node in the tree. This is done by a run of a *modified ddoc* algorithm. A node in the tree to which the delay bounded path has the minimum cost is taken as the destination in the tree, the source being the new node. The source node is joined to the destination node in the tree in a similar way as follows. The cost and delays of minimum delay path and Δ -bounded minimum cost path are found. A cost and delay factor are computed. If the cost factor outweighs the delay factor, then we find a minimum cost path bounded by the average of the delays of the two paths. Otherwise, if cost factor is not too high, we use minimum delay path. This algorithm prevents cycles as a new path is being added and gives the minimum incremental network cost. Thus we call it a best effort heuristic. Also, the resulting tree formed is guaranteed to have bounded delay between any pair of multicast nodes. One disadvantage of this algorithm is that the order in which we join the nodes to the tree matters.

2 Simulation and Testing

Undirected, connected, random graphs were used to simulate networks in the experiments. The experiments were run for different network sizes using a fixed average node degree of 4. For each network size, the positions of the nodes were fixed as in a *hypercube-type* topology. The links (edges in the graph) connecting these nodes were generated using a random link generator. In terms of distance, the network spanned an area of $3000 \times 2400 \text{ km}^2$ ($0.015 \times 0.012 \text{ seconds}^2$ in terms of propagation delay). The total maximum link bandwidth was taken to be 155 Mbps. The cost of a link was a dynamic function of the available link bandwidth. The links (bandwidth) were to be shared among the traffic streams from all multicast group members.

Figure 5 shows the probability of member success of the different algorithms as a function of the group size for delay constraint $\Delta = 35\text{ms}$ and network size = 32 nodes.

Figure 6 shows the comparison of the cost of the multicast trees generated by the algorithms against cost of the delay constrained *minimumcost* tree for a network size of 32 nodes and varying multicast group size. The delay tolerance Δ is taken to be 35ms. For purposes of visual clarity, we present Salaama's most cost-optimal algorithm (DCINITIAL) and BERD.

Figure 7 shows the probability of member success as a function of the delay tolerance Δ . We performed this experiment for a network of size 64 nodes, and a multicast group size of 16 nodes. Amongst Salaama's algorithms, MinMaxD chooses the node

with least maximum delay to any group member to be the center and hence increases the probability that *DCSHARED* succeeds. For the network that we have chosen ($15ms^2 \times 12ms^2$ in terms of delay), the range of delays between 15ns and 25ns is the only meaningful range. The values of delay below this range are too small to support any significant group sizes. Above this range, the delay values become quite large so as to no longer act as a major constraint for the formation of a multicast tree. Moreover, typical multimedia applications also require a delay constraint in this range. In this region, BERD performs better than any other algorithm.

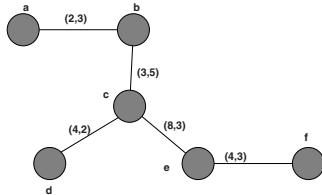
Figure 8 shows the number of admitted multicast sessions as a function of the multicast group size for a 16 node network. The experiment was run for different delay constraint settings. The delay constraint Δ is 25ms for Fig. 8 (A) and 35 ms for Fig. 8 (B). We consider a multicast session to be successful if at least half the members are able to join and form a shared tree.

3 Conclusions

Both BERD and DCBH avoid star-like topology of the resulting shared tree. BERD has no concept of a center node. Although DCBH does keep a center, it updates the center whenever the shared tree changes as a consequence of a new member joining the group. This frees us from the requirements of a fast switching center node. The simulation results show that BERD outperforms all the other algorithms. The probability of member success of BERD or DCBH is very high as compared to existing algorithms since BERD and DCBH conserve on the delay available for joining to the existing shared tree, without any significant increase in cost. BERD also gives low cost of the multicast tree obtained. In fact the cost of the tree formed using BERD is only marginally higher as compared to the cost of the tree formed by the optimal algorithm. This is so because BERD tries to join at that node which gives the minimal cost delay bounded path of all the nodes present in the existing shared tree. The two algorithms distribute the load on the network uniformly. This is because of the structure of trees formed by these, as described above. In case of DCBH, the center is updated as soon as a new member joins the tree. This avoids the problems of a static center and hence distributes the load very well.

References

1. R. Bajaj, C.P. Ravikumar and S. Chandra. Distributed delay-constrained multicast path setup algorithm for high-speed networks. Proceedings of International Conference on High Performance Computing. 1997. Pages 438–442.
2. H.F. Salaama. *Multicast Routing for Real-time Communication on High-Speed Networks*. Ph.D. Thesis. North Carolina State University, Raleigh, USA. 1996.
3. H.F. Salaama et al. *Delay-constrained Shared Multicast Trees*. Tech. Report, Department of Electrical and Comp. Engg., North Carolina State Univ., USA. 1997.
4. G.Kumar, N.Narang and C.P.Ravikumar. *Efficient Algorithms for Delay-bounded Minimum Cost Path Problem in Communication Networks*. In Proceedings of the Fifth International Conference on High Performance computing. 1998.

Fig. 1: Illustration of Residual Delay. $\Delta = 12$.

```
procedure DCBH( $M, \Delta$ )
```

```

 $T \leftarrow \text{NULL}$  // Partial Tree
for each ( $n \in M$ ) begin
     $center \leftarrow \text{findCenter}(M)$ 
    if (( $Path_A \leftarrow \text{dboc}(M, center)$ )
         $\neq \text{NULL}$ ) begin
         $Path_B \leftarrow \text{Dijkstra}(n, center)$ 
        //  $B$  has minimum delay from  $n$  to  $center$ 
        if ( $\text{Delay\_Factor}(Path_A, Path_B)$ 
            outweighs
            ( $\text{Cost\_Factor}(Path_A, Path_B)$ )
        then
             $\text{AddPath}(Path_B, T)$ 
        else begin
             $delay_{new} \leftarrow \text{DelayFunction}(A, B)$ 
            // A delay between those of  $A, B$ 
             $Path_C \leftarrow \text{dboc}(n, center, delay_{new})$ 
             $\text{AddPath}(Path_C, T)$ 
        end
         $center \leftarrow \text{updateCenter}(T)$ 
    end
    else begin
         $\text{tryAllNodes}(n, T, \Delta)$ 
        // Refer to Figure 3
         $center \leftarrow \text{updateCenter}(T)$ 
    end
end
```

Fig. 2: Algorithm DCBH

```
procedure tryAllNodes( $n, T, \Delta$ )
```

```

 $PathSet \leftarrow \text{mdboc}(n, T, \Delta)$ 
// Each path  $n \dots d$ ,  $d \in T$ , is
// bounded by residual delay at  $d$ 
 $Path_A \leftarrow \text{SelectBestPath}(PathSet)$ 
// The last node in the path is the node
// in  $T$  to which  $n$  must join
 $d \leftarrow Path_A.lastNode$ 
 $Path_B \leftarrow \text{Dijkstra}(n, d)$ 
// Minimum delay path from  $n$  to  $d$ 
if ( $\text{Delay\_Factor}(Path_A, Path_B)$ 
    outweighs
    ( $\text{Cost\_Factor}(Path_A, Path_B)$ )
then
     $\text{AddPath}(Path_B, T)$ 
else begin
     $delay_{new} \leftarrow \text{DelayFunction}(A, B)$ 
     $Path_C \leftarrow \text{dboc}(n, d, delay_{new})$ 
     $\text{AddPath}(Path_C, T)$ 
end
```

Fig. 3: Procedure tryAllNodes()

```
procedure BERD( $M, \Delta$ )
```

```

 $T \leftarrow \text{NULL}$ 
for each ( $n \in M$ ) begin
     $PathSet \leftarrow \text{mdboc}(n, T, \Delta)$ 
     $Path_A \leftarrow \text{SelectBestPath}(PathSet)$ 
     $d \leftarrow Path_A.lastNode$ 
     $Path_B \leftarrow \text{Dijkstra}(n, d)$ 
    if ( $\text{Delay\_Factor}(Path_A, Path_B)$ 
        outweighs
        ( $\text{Cost\_Factor}(Path_A, Path_B)$ )
    then
         $\text{AddPath}(Path_B, T)$ 
    else begin
         $delay_{new} \leftarrow \text{DelayFunction}(A, B)$ 
         $Path_C \leftarrow \text{dboc}(n, d, delay_{new})$ 
         $\text{AddPath}(Path_C, T)$ 
    end
end
```

Fig. 4: Algorithm BERD

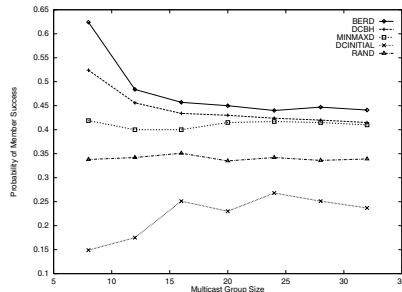
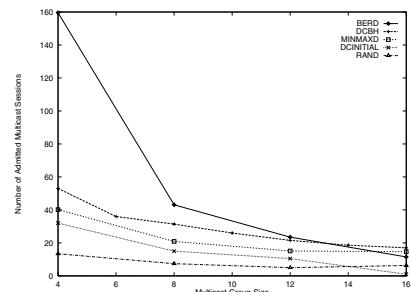


Fig. 5: Member Success Prob. Vs Group Size.
32 nodes, $\Delta = 35ms$.



(A)

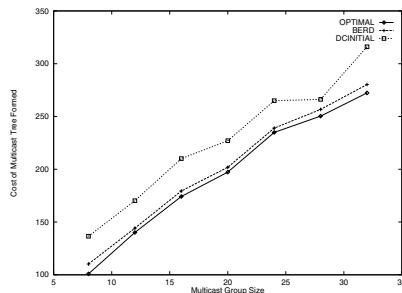
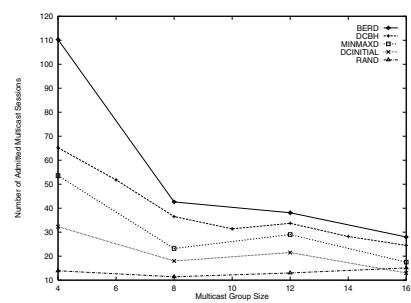


Fig. 6: Cost of Multicast Trees Vs Multicast Group Size. $\Delta = 35ms$, 32 nodes.



(B)

Fig. 8: Number of Admitted Sessions Vs Multicast Group Size. 16 nodes. (A) $\Delta = 25$.
(B) $\Delta = 35$.

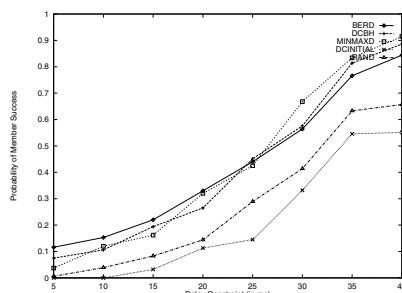


Fig. 7: Probability of Member Success Vs Δ .
64 nodes, group of 16.

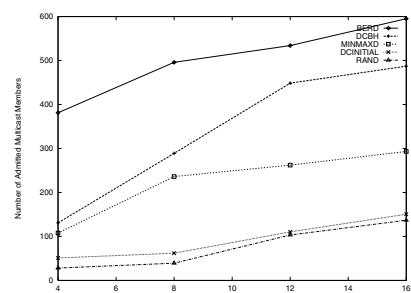


Fig. 9: Number of Admitted Members Vs Multicast Size. 16 nodes. $\Delta = 35ms$

Panel

Whither Indian Computer Science R&D?

Moderator

Sriram Vajapeyam, Indian Institute of Science

Panelists

Alok Aggarwal, IBM India Solutions Research Centre

R.K. Arora, Centre for Development of Advanced Computing, Bangalore

Arvind, MIT

Kris S. Gopalakrishnan, Infosys

Pankaj Jalote, Indian Institute of Technology, Kanpur

Ashok Jhunjhunwala, Indian Institute of Technology, Chennai

Krithi Ramamirtham, Univ. of Massachusetts, Amherst and Indian

Institute of Technology, Mumbai

M. Vidyasagar, Centre for Artificial Intelligence and Robotics (CAIR),

India

Mini Symposium

High Performance Data Mining

Organizers:

Vipin Kumar

University of Minnesota

Jaideep Srivastava

University of Minnesota

The current decade has seen an explosive growth in database technology and the amount of data collected. This has created an unprecedented opportunity for "data mining," which is a process of efficient supervised or unsupervised discovery of interesting information hidden in the data. The focus of this symposium is to bring together experts from the academia and the industry, including researchers and practitioners, to discuss issues in this important area. Owing to the huge size of data and computation involved in data mining algorithms, parallel processing is often considered an essential component for a successful data mining solution - which will be a special emphasis of the symposium. Talks in this mini-symposium will cover both the current state-of-the-practice as well as state-of-the-research in data mining. A highlight of the symposium is a panel discussion on the theory and practice of data mining.

Session III-A

Parallel Algorithms - I
Chair: Amar Mukherjee
University of Central Florida

Self-Stabilizing Network Decomposition

Fatima Belkouch¹, Marc Bui², Liming Chen³, and Ajoy K. Datta^{4*}

¹ Heudiasyc, Université de Technology de Compiègne, France

² Laboratoire de Recherche LRIA, Université Paris 8, France

³ ICTT, Département Maths/Info., Ecole Centrale de Lyon, France

⁴ Department of Computer Science, University of Nevada, Las Vegas

*Supported by a sabbatical leave grant from the University of Nevada, Las Vegas.

Abstract. We present a simple and efficient self-stabilizing protocol for the network partitioning problem. Given a graph with k^2 nodes, our decomposition scheme partitions the network into connected and disjoint partitions, with k nodes per partition. The proposed algorithm starts with a spanning tree of the graph, but uses some links which do not belong to the tree, if necessary. The protocol stabilizes in $(3h + 1)$ steps, where h is the height of the tree, and adapts to the dynamic configuration of the network.

1 Introduction

As networks grow larger, the control and communication protocols become more complex and inefficient. The motivation for decomposing (large) networks is to improve the performance of the protocols, i.e., avoid the performance degradation due to the excessive growth of the network. We propose the first *self-stabilizing* network partitioning protocol. In the following, we will discuss the related work in the area of network decomposition and then present our contributions.

1.1 Related Work

The concept of network decomposition was introduced in [2]. A fast algorithm is proposed to partition a network with $O(n^\epsilon)$ diameter clusters, where $\epsilon = O(\sqrt{\log \log n} / \sqrt{\log n})$. This algorithm requires $O(n^\epsilon)$ time. The algorithms in [3] and [6] are improved versions of the scheme proposed in [2] in terms of the quality of the decomposition. However, these algorithms are inherently sequential and their distributed implementation requires $O(n \cdot \log n)$ time. Another method, called *block decomposition*, is introduced in [7]. In this algorithm, the nodes of the graph are partitioned into blocks which may not create connected subgraphs. The algorithm requires both the number of blocks and the diameter of the connected blocks to be small. Some randomized distributed partitioning algorithms exist in the literature. The algorithm in [6] achieves a high quality network decomposition with a high probability by introducing randomization, and is very efficient in time. An asynchronous randomized algorithm with $O(\log^2 n)$ time and

$O(|E| \cdot \log^2 n)$ communication complexity is presented in [7]. A deterministic sub-linear time distributed algorithm for network decomposition has been presented recently in [1]. The algorithm takes $O(n^\epsilon)$ time, where $\epsilon = O(1/\sqrt{\log n})$. This paper also includes a randomized algorithm for a high quality network decomposition in polylogarithmic expected time. The protocol is based on a restricted model, the *static synchronous network*. In such a model, the communication is assumed to be completely synchronous and reliable, there is no limit on the size of the messages, and all operations within a subgraph of diameter t are performed centrally by collecting all the information at the leader. Then the leader computes the partition locally. Therefore the time for running the algorithm increases by a factor of t . This method will also need a leader election algorithm. In summary, all previous approaches fell short of designing a distributed, deterministic, and fault-tolerant network partitioning scheme.

1.2 Contributions

We present a simple, efficient, distributed, deterministic, and self-stabilizing protocol for constructing network partitions. An important application (described later) of our method is the construction of quorum systems. The key contribution of our work is in designing a computationally inexpensive self-stabilizing partitioning protocol which ensures a high quality of quorums in terms of response time and site load in the system. We combine the basic network partitioning protocol with a *Propagation of Information With Feedback* scheme [5] to design a self-stabilizing partitioning protocol which stabilizes in $(3h + 1)$ time. Since our algorithm is self-stabilizing, the protocol will re-compute the partitions in the event of topological changes.

1.3 Outline of the Paper

In Section 2, we describe the distributed systems and model we consider in the paper. In Section 3, we specify the problem of network partitioning. In Section 4, we present the basic idea of the network partitioning strategy and then, present the partitioning algorithm. The idea of the self-stabilizing partitioning scheme and its correctness is given in Section 5. In Section 6, we explain how to maintain the properties of the partitions in spite of a dynamic change in the network and present the quorum application.

2 Preliminaries

In this section, we define the distributed systems and programs considered in this paper, and state what it means for a protocol to be self-stabilizing.

System. A *distributed system* is an undirected connected graph, $S = (V, E)$, where V is a set of nodes ($|V| = n$) and E is the set of edges. Nodes represent *processors* (denoted by p) and edges represent *bidirectional communication links*. We consider arbitrary rooted asynchronous networks. We then assume an

underlying BFS spanning tree protocol. Each processor p maintains its set of neighbors, denoted as N_p .

Programs. Every processor, except the leaf processors, executes the same program. This type of programs is known as a *semi-uniform distributed algorithm*. The program consists of a set of *shared variables* and a finite set of actions. A processor can only write to its own variables and can only read its own variables and variables owned by the neighboring processors. Each action is identified by a label and has the following form: $< \text{label} > :: < \text{guard} > \longrightarrow < \text{statement} >$ The guard of an action in the program of p is a boolean expression involving the variables of p and its neighbors. The statement of an action of p updates one or more variables of p . An action can be executed only if its guard evaluates to true. We assume that the actions are atomically executed : the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step. The atomic execution of an action of p is called a *step* of p . In the sequel, we refer to the state of a processor and system as a (*local*) *state* and *configuration*, respectively. Let a distributed protocol \mathcal{P} be a collection of binary transition relations denoted by \mapsto , on \mathcal{C} , the set of all possible configurations of the system. A *computation* of a protocol \mathcal{P} is a *maximal* sequence of configurations $e = (\gamma_0, \gamma_1, \dots, \gamma_i, \gamma_{i+1}, \dots)$, such that for $i \geq 0$, $\gamma_i \mapsto \gamma_{i+1}$ (a single *computation step*) if γ_{i+1} exists, or γ_i is a terminal configuration. During a computation step, one or more processors execute a step and a processor may take at most one step. This execution model is known as the *distributed daemon*. We use the notation $\text{Enable}(A, p, \gamma)$ to indicate that the guard of the action A is true at processor p in the configuration γ , and p is said to be *enabled*. We assume a *weakly fair* daemon, meaning that if a processor p is continuously *enabled*, p will be eventually chosen by the daemon to execute an action. The set of computations of a protocol \mathcal{P} in system S starting with a particular configuration $\alpha \in \mathcal{C}$ is denoted by \mathcal{E}_α . The set of all possible computations of \mathcal{P} in system S is denoted as \mathcal{E} .

A configuration β is *reachable* from α , denoted as $\alpha \leadsto \beta$, if there exists a computation $e = (\gamma_0, \gamma_1, \dots, \gamma_i, \gamma_{i+1}, \dots) \in \mathcal{E}_\alpha(\alpha = \gamma_0)$ such that $\beta = \gamma_i(i \geq 0)$.

Predicates. Let \mathcal{X} be a set. $x \vdash P$ means that an element $x \in \mathcal{X}$ satisfies the predicate P defined on the set \mathcal{X} . A predicate is non-empty if there exists at least one element that satisfies the predicate. We define a special predicate *true* as follows: *for any* $x \in \mathcal{X}$, $x \vdash \text{true}$.

Self-Stabilization. We use the following term, *attractor* in the definition of self-stabilization.

Definition 1 (Attractor). Let X and Y be two predicates of a protocol \mathcal{P} defined on \mathcal{C} of system S . Y is an *attractor* for X if and only if : $\forall \alpha \vdash X : \forall e \in \mathcal{E}_\alpha : e = (\gamma_0, \gamma_1, \dots) :: \exists i \geq 0, \forall j \geq i, \gamma_j \vdash Y$. We denote this relation as $X \triangleright Y$.

Definition 2 (Self-Stabilization). The protocol \mathcal{P} is *self-stabilizing* for the specification $\text{SP}_\mathcal{P}$ on \mathcal{E} if and only if there exists a predicate $L_\mathcal{P}$ (called the *legitimacy predicate*) defined on \mathcal{C} such that the following conditions hold:

1. $\forall \alpha \vdash L_\mathcal{P} : \forall e \in \mathcal{E}_\alpha :: e \vdash \text{SP}_\mathcal{P}$ (*correctness*).
2. $\text{true} \triangleright L_\mathcal{P}$ (*closure and convergence*).

3 Network Partitioning

The network partitioning problem deals with the grouping of V into k connected and disjoint partitions, each partition consisting of k nodes. We use $G[S]$ to denote the subgraph induced by the subset $S \subset V$ in G . A partitioned network, denoted as $P = \{B_1, B_2, \dots, B_k\}$, is a collection of k partitions such that the following conditions are true:

$$\forall i, j \in \{1..k\} :: \quad \begin{aligned} 1) \quad & G[B_i] \text{ is connected}, & 2) \quad & |B_i| = k, \\ 3) \quad & \cup_i B_i = V, & 4) \quad & B_i \cap B_j = \emptyset, i \neq j \end{aligned}$$

A partition B_i is called a *complete partition* if $|B_i| = \sqrt{n}$. If $|B_i| < \sqrt{n}$, then the partition is considered as a *non-complete partition*. Connected and disjoint partitions are called *correct partitions*. A partition B_i , constructed at processor p , is characterized by *Node_list* the list of nodes in B_i and *External Links*, a set of edges defined by $L = \{(x, y) \in G_i | x \in B_i, y \notin B_i, y \text{ is a descendant of } p\}$

Specification of the Partitioning problem. The problem solved in this paper is to design a deterministic self-stabilizing algorithm for decomposing a graph $G(V, E)$ ($|V| = n = k^2$) into k connected and disjoint partitions, $P = \{B_1, B_2, \dots, B_k\}$, each partition consisting of k nodes. The resulting partitions must be known to all processors in the network. The algorithm we propose in this paper computes partitions which satisfy the above properties, only if such a decomposition exists for the input graph. The characteristics of a graph that can be decomposed into this kind of partitions is still an open question.

4 Network Partitioning Algorithm \mathcal{NP}

The algorithm uses a spanning tree of the underlying network as the input. The basic scheme is as follows: The decomposition process starts from the leaf processors. Every leaf processor forms a non-complete partition. In each step, some nodes participate in the algorithm to create larger partitions until we obtain a set of complete partitions which cover the whole network. Eventually, this process reaches the root. The root now knows the partitions and so, broadcasts the information about the partitions to all processors in the network.

Each processor p uses two variables: LD_p and GD_p to implement the local and global decomposition, respectively. LD_p contains the set of partitions that covers the subtree T_p rooted at p . GD_p consists of the set of partitions created at the root, p_0 , and is broadcast to other processors. GD_p covers the whole graph, and is the final and correct decomposition. The partitioning algorithm shown in Algorithm 1 contains some *macros*, *predicates*, and *actions*. The macros are not variables and they are dynamically evaluated. The predicates are used to describe the guards of the actions in Algorithm 1.

Algorithm 1 (\mathcal{NP}) Distributed Algorithm for Network Partitioning at p **Variables**
 $LD_p = \{B_1^p, B_2^p, \dots, B_k^p\}$ /* Local Decomposition */
 $GD_p = \{B_1^p, B_2^p, \dots, B_k^p\}$ /* Global Decomposition */
Macro
 $Parent_p = p$'s parent in the spanning tree

 $Chi_p = \{q \in N_p, Parent_q = p\}$ p 's children

 $|B_i^q| = |B_i^q.Nodes.List|$ Number of nodes in the partition B_i^q
 $NCP_p = \{B \in \bigcup_{q \in Chi_p} LD_q, |B| < \sqrt{n}\}$ Set of non-complete partitions of all children of p
 $CP_p = (\bigcup_{q \in Chi_p} LD_q) \setminus NCP_p$ Set of complete partitions of all children of p
 $NCP_p^+ = \{x : \exists B \in NCP_p \text{ st. } x \in B\}$ Set of nodes in NCP_p
Predicates
 $Leaf_p \equiv (Chi_p = \emptyset)$
 $Path(i, j, S) \equiv (\exists k \in S : (i, k) \in G(S) \wedge Path(k, j, S)) \vee (i = j)$
 $Connected(S) \equiv (\forall i \in S, \forall j \in S : Path(i, j, S))$
 $Combine_p \equiv (|NCP_p^+| = \sqrt{n}) \wedge (Connected(NCP_p^+))$
 $Reconstruct_p \equiv ((|NCP_p^+| > \sqrt{n})) \vee (|NCP_p^+| = \sqrt{n} \wedge \neg Connected(NCP_p^+))$
Actions
 $/* Leaf processors */$
 $A1 :: Leaf_p \rightarrow LD_p := \{\{p\}\}; GD_p := GD_{parent_p}$
 $/* Other processors */$
 $A2 :: (|NCP_p^+| < \sqrt{n}) \rightarrow LD_p := CP_p \cup \{NCP_p^+ \cup \{p\}\}; GD_p := GD_{parent_p}$
 $A3 :: NCP_p^+ = \emptyset \rightarrow LD_p := CP_p \cup \{\{p\}\}; GD_p := GD_{parent_p}$
 $A4 :: Combine_p \rightarrow LD_p := CP_p \cup \{NCP_p^+\} \cup \{\{p\}\}; GD_p := GD_{parent_p}$
 $A5 :: Reconstruct_p \rightarrow Combine - Partitions(\bigcup_{q \in Chi_p} LD_q, A_{nc}, A_c)$

if ($Connected(A_{nc} \cup \{p\})$) and ($|A_{nc}| < \sqrt{n}$)
then $LD_p := A_c \cup \{A_{nc} \cup \{p\}\}$
else $LD_p := \bigcup_{q \in Chi_p} LD_q \cup \{\{p\}\}$ /* Notable */
 $GD_p := GD_{parent_p}$

Completing the Non-Complete Partitions : We can easily see from Algorithm \mathcal{NP} that the basic scheme to construct partitions, without any composition of partitions, produces correct partitions (as per Conditions 1 – 4 in Section 3). So, we now need to show that Procedure *Combine-Partitions* (Action A5) also combines the non-complete partitions and creates correct complete partitions. It uses two Procedures called *Complete* and *More-Links*:

Input to Procedure *Combine-Partitions*: A set of correct partitions $D = \{B_1^0, B_2^0, \dots, B_i^0, \dots, B_k^0\}$. The subset $\{B_1^0, B_2^0, \dots, B_i^0\}$ consists of the non-complete partitions (NCP), and the subset $\{B_{i+1}^0, B_{i+2}^0, \dots, B_k^0\}$ is the set of complete partitions (CP).

Output of Procedure *Combine-Partitions*: It returns a set, A_c , of the largest set of complete partitions that can be constructed at this processor, and a set, A_{nc} , of the rest of nodes.

Algorithm 2 (PC) Procedure Complete

1. **Predicate**
2. $\text{FoundLink}(l_k, Pt, v_r) \equiv (l_k = (e_k, f_k) \in E) \wedge (e_k \in Pt) \wedge (f_k \in v_r)$
 $\wedge (f_k \notin Pt) \wedge (|Pt \cup \{f_k\} \cup \{\text{Descendant}(f_k, v_r)\}| = \sqrt{n})$
3. $\text{New}(i, B_{nc}, B_c, j) \equiv (\forall k < j, \text{Complete}(i, B_{nc}, B_c, k) \notin \text{SavedCalls})$
4. **Macro**
5. $\text{Descendant}(p, Pt)$ = The set of nodes that belongs to T_p in partition Pt
6. $\text{CurrentPartition}(Pt, f_k) = Pt \cup \{f_k\} \cup \{\text{Descendant}(f_k, v_r)\}, (f_k \in v_r)$
7. SavedCalls = The past calls of Procedure *Complete*
8. Every call is stored as: $\text{Complete}(i, B_{nc}, B_c, j)$
9. **Procedure Complete** (i, B_{nc}, B_c, j)
10. if $|B_{nc}| > 1$ then
11. select v_i from B_{nc}
12. $Pt = v_i$
13. if $\text{FoundLink}(l_k, Pt, v_r)$ then $Pt = \text{CurrentPartition}(Pt, k)$
14. else More-Links (1, L, Pt) /* L : External Link of Pt */
15. if $(|Pt| = \sqrt{n})$ then
16. update B_{nc}
17. update B_c
18. if $(\text{New}(1, B_{nc}, B_c, j))$ then $\text{Complete}(1, B_{nc}, B_c, j++)$
19. else exit /* Cannot be completed using this level */
20. /* A cycle problem. Try at a higher level. */
21. else
22. if $((i < |B_{nc}|) \text{ and } \text{New}(i++, B_{nc}, B_c, j))$ then $\text{Complete}(i, B_{nc}, B_c, j)$
23. else exit /* Cannot be completed using this level. */
24. else /* success i.e. all partitions in NCP are completed */

Algorithm 3 (PML) Procedure More-Links

1. **Procedure More-Links** (i, L, Pt)
2. if $i \leq |L|$ then take $l_i = (e_i, f_i)$ from L
3. if $|\text{CurrentPartition}(Pt, f_i)| = \sqrt{n}$ then
4. $Pt = \text{CurrentPartition}(Pt, f_i)$ exit
5. else if $|\text{CurrentPartition}(Pt, f_i)| < \sqrt{n}$ then
6. $Pt = Pt \cup \{f_i\} \cup \{\text{Descendant}(f_i, v_r)\}$
7. $L = \text{ExternalLink of } Pt$
8. More-Links (1, L, Pt)
9. else More-Links ($i + 1, L, Pt$)
10. end

Algorithm 4 (PCP) Procedure Combine-Partitions

1. **Procedure Combine-Partitions** (D, A_{nc}, A_c)
2. $i = 1$ /* Tries to complete the first non-complete partition. */
3. $j = 1$ /* j is used to detect a cycle which can occur when *Complete* is called twice with the same parameters. */
4. $\text{Complete}(i, B_{nc}, B_c, j)$ /* $D = B_{nc} \cup B_c$ */
5. $A_{nc} = B_{nc}$ $A_c = B_c$

Property 1. Procedure Combine-Partitions computes in step m , a set of correct partitions $\{B_1^m, B_2^m, \dots, B_k^m\}$ such that, $\exists j, 1 \leq \forall i \leq k :$
 i) $|B_i^m| = \sqrt{n}$ for $i < j$, ii) $|B_i^m| \leq \sqrt{n}$ for $i = j$ and iii) $|B_i^m| = 0$ for $i > j$

5 Partitioning using the PFC scheme

First, let us quickly review the well-known *PIF scheme* on tree structured networks. The PIF scheme is the repetition of a *PIF cycle*. This one can be informally defined as follows: The root initiates the *broadcast* phase and its descendants forward the broadcast message to their descendants. This phase terminates at the leaf processors by initiating the *feedback* phase. Once the feedback message reaches the the root, the current PIF cycle terminates. In the PFC scheme [5], *Propagation of Information With Feedback and Cleaning* an extra phase, called the *cleaning* phase [8], is added to clean the trace of the previous PIF cycle. In the PFC scheme, the normal PIF cycle is guaranteed to start by 1 step. As mentioned in Section 4, the process of partitioning will complete at the root. Then this information must be propagated to all nodes in the network. We use the PFC scheme of [5] to implement this. After the root starts a normal PFC cycle, the root will propagate the partitions in the *broadcast* phase. But, since the system may start in an incorrect configuration, this information may be incorrect. But, now, at the termination of the *broadcast* phase, the *feedback* phase will collect the correct information at the root. So, the next *broadcast* phase is guaranteed to deliver the correct partitioning information to all the nodes in the network.

Theorem 1. *Algorithm \mathcal{NP} combined with the PFC scheme is a self-stabilizing network partitioning scheme.*

Proof. (Correctness) Follows from the construction of the partitions in the algorithm and also the description of the process in Section 4.

(Closure) Follows from the same property of the PFC scheme (see Section 4).

(Convergence) We need to show that starting from an arbitrary partitioning $P = \{B_1^0, B_2^0, \dots, B_i^0, \dots, B_k^0\}$, after a finite number of steps, the partitions will be correct. Using the PFC algorithm, we ensure that all processors will receive the correct partitioning information in at most $1 + h + h + h = 3h + 1$ steps. So, Algorithm \mathcal{NP} takes $3h + 1$ steps to stabilize.

6 Discussions

Since the topology of the network may change over time, it is necessary to be able to maintain the legal partitions when the network size becomes $n \neq k^2$. Theoretically, to obtain the best decomposition of the network, we distinguish two cases where ($k^2 + 1 \leq n \leq k^2 + 2k$): i) if $n = k^2 + k_0$ and $0 \leq k_0 \leq k$, we construct k_0 partitions with $k + 1$ nodes each and $k - k_0$ partitions with k nodes each. ii) if $n = k^2 + k + k_0$ and $k_0 \geq 0$, we construct k_0 partitions with $(k + 1)$

nodes and $(k - k_0 + 1)$ partitions with k nodes. Decomposing the network in this manner offers is efficient but its implementation is difficult in a distributed context. Each process must know the current value of n , the number of partitions, and the number of nodes it must include in each partition. One trivial strategy for handling both the addition and removal of a node is to construct k partitions with k nodes in each partition and one partition with $(n - k^2)$ nodes. The non-complete partition is constructed by the root and all other nodes maintain the same behavior. The worst case of this solution leads to a decomposition with a partition of one node (the root). The last method adapts to the dynamic network but does not guarantee the quality of the decomposition.

The decomposition scheme serves for constructing quorums for general large networks. This structure provides an important and reliable tool for various applications in distributed systems. A *quorum set* $Q = \{q_1, q_2, \dots, q_k\}$ under a set V of elements is a non-empty set where each element is a non-empty subset of V and $q_i \not\subseteq q_j$, for all i, j . A *complementary quorum set* $Q^c = \{q^c_1, q^c_2, \dots, q^c_k\}$ is an other quorum set under V such that $q^c_i \cap q_j \neq \emptyset$. The pair $B = (Q, Q^c)$ is called a *bicoterie*. We define a quorum q in Q as the set of all nodes in one partition and a quorum q^c in Q^c as a set formed by one node from every partition. This construction guarantees a low communication cost and a small and balanced load [4].

References

- [1] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast distributed network decomposition and covers. *Journal of Parallel and Distributed Computing*, 32(2):105–114, 1996.
- [2] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. of the 30th IEEE Symposium on Foundations of Computer Science*, pages 364–369, 1990.
- [3] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. of the 31th IEEE Symposium on Foundations of Computer Science*, pages 503–513, 1990.
- [4] F. Belkouch, L. Chen, and A. Elfatni. Routage sémantique basé sur les quorums. *Calculateurs Parallèles, Numéro thématique : Routage dans les Réseaux*, 11(ISBN-2-7462-0040-6):87–101, 1999.
- [5] A. Bui, AK. Datta, F. Petit, and V. Villain. State-optimal snap-stabilizing PIF in tree networks. In *Proc. of the Fourth Workshop on Self-Stabilizing Systems*, pages 78–85. IEEE Computer Society Press, 1999.
- [6] N. Linial and M. Saks. Decomposing graphs into region of small diameter. In *Proc. of the Second Annual ACM-SIAM, Sympsium on Discrete Algorithms*, pages 320–330, 1991.
- [7] N. Linial and M. Saks. Low diameter graph decomposition. *Combinatoria*, 13:441–454, 1993.
- [8] V. Villain. A new lower bound for self-stabilizing mutual exclusion algorithms. Technical Report RR98-08, LaRIA, University of Picardie Jules Verne, 1998, also, presented at Dagstuhl Workshop on Self-Stabilization, August 17-21, 1998, Germany.

Performance Analysis of a Parallel PCS Network Simulator^{*}

Azzedine Boukerche¹, Sajal K. Das², Alessandro Fabbri¹, and Oktay Yildiz¹

¹ Department of Computer Science, University of North Texas
Denton, TX 76203-1366, USA
`{boukerche, fabbri, yildiz}@cs.unt.edu`

² Center for Research in Wireless Computing (CReW)
Department of Computer Science & Engineering, University of Texas at Arlington
Arlington, TX 76019-0015, USA
`das@cse.uta.edu`

Abstract. This paper presents an analytical performance study of a parallel simulation testbed for PCS networks, called *Simulator of Wireless Mobile Networks* (SWiMNet), and reports experimental results obtained using a realistic model executed on a cluster of workstations. SWiMNet achieves linear speedup, thereby significantly reducing the execution time of a PCS network simulation. Performance results demonstrate that our simulation model is consistent with the analytical study.

1 Introduction

Rapid development in portable computing platforms and wireless communication technology has led to significant interest in mobile computing and wireless networking. One such technology is called *personal communication system* (PCS). Mathematical analysis has brought some insight into the design of PCS networks, but analytical methods are often not general or sufficiently detailed. Therefore, it is through simulation that telecommunication system engineers can obtain crucial performance characteristics.

With increasing use of PCS simulation to design large and complex systems, PCS systems have brought several challenges to wireless simulation and *parallel discrete event simulation* (PDES) communities in general [4]. These challenges require not only extension and advances in current parallel simulation methodologies, but also innovative techniques to deal with the rapidly expanding features of PCS systems. Several results have been reported in the literature on parallel simulation of PCS networks [2,7,8,9,10,12]. To the best of our knowledge, these approaches do not consider realistic PCS models, the performances of which strongly depend on both mobile host (MH) profiles and PCS coverage area characteristics. We believe real PCS models must be considered to make meaningful recommendations to system designers.

Recently, we designed and implemented a parallel simulator for large scale PCS networks, called *SWiMNet* (Simulator of Wireless Mobile Networks, [1]). It exploits event precomputation made possible by model independence among realistic PCS model components. Process coordination is based on a hybrid of

* This work is supported by Texas Advanced Research Program under Award No. TARP-97-003594-013, and by a grant from Nortel Networks, Richardson, Texas.

both *conservative* [3] and *optimistic* [5] PDES schemes. In this paper, we present an analytical performance study of SWiMNet to predict the achievable speed-up.

The paper is organized as follows. Section 2 describes the basics of PCS networks. Section 3 summarizes our methodology to simulate PCS networks. An analytical study of the model is presented in Section 4. Section 5 presents experimental results, while Section 6 concludes the paper.

2 PCS Network Basics

A *personal communication system* (PCS) is a network that provides low-power and high-quality wireless communication to *mobile hosts* (MHs) [6,11]. The coverage area is partitioned into *cells*, each serviced by an antenna or *base station* (BS). Cell sizes and shapes depend on signal strengths and obstacles to signal propagation. *Radio channels* are allocated by a BS for calls to/from MHs within its own cell. Channels can be assigned to cells according to a fixed channel assignment (FCA) scheme or a dynamic channel assignment (DCA) scheme [6,11], or a combination thereof. In FCA, each BS is statically assigned a set of channels, while in DCA, channels are dynamically assigned to cells. In this paper, we consider simulation of FCA schemes.

When a call attempt is made to/from an MH in a cell where there are no channels available, the call is *blocked*. If an MH moves from one cell to another while a call is in progress, the MH must be connected to the destination BS (a *hand-off*). The original BS releases the channel used by the MH, while the destination cell tries to allocate a channel for the call to continue. If an ongoing call cannot be handed-off due to lack of available channels, the call is *dropped*. An important performance criteria for a PCS network is *call blocking probability*, defined as the ratio of blocked calls to attempted calls. The blocking probability should be kept small, typically around 1% [2].

3 SWiMNet

In [1], we proposed the *Simulator of Wireless Mobile Networks* which exploits event precomputation due to the following assumption: *mobility and call arrival are independent of the state of the PCS network, and they are independent of each other*. As shown in Fig. 1, processes composing SWiMNet are grouped into two *stages*: a *precomputation stage* (Stage 1), and a *simulation stage* (Stage 2). Processes in Stage 1 precompute all possible events for all MHs assuming all channel requests are satisfied. Possible events are: (1) *call_arrival*, (2) *call_termination*, (3) *move_in*, and (4) *move_out*. Events relative to different MHs can be precomputed independently, therefore Stage 1 processes do not communicate. Precomputed events are sent to Stage 2, where their occurrence is checked according to the state of the PCS network simulation. Stage 2 processes cancel events relative to calls which turn out to be blocked or dropped due to channel unavailability. Since events for the same call can span various cells, Stage 2 processes need to notify other processes of blocked calls.

Process coordination in SWiMNet is based on a *hybrid* approach, using both conservative and optimistic techniques. A conservative scheme is used for communication from Stage 1 to Stage 2 such that Stage 1 processes send precomputed

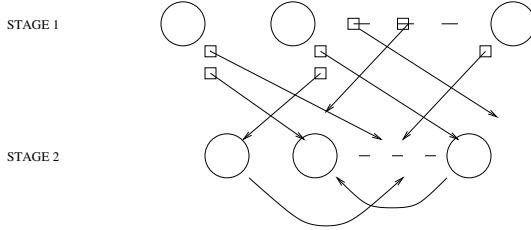


Fig. 1. The SWiMNet structure

events in occurrence time order, and periodically broadcast their local simulation time to Stage 2 processes through *null messages*. Stage 2 processes can consequently establish which precomputed events are safe to be simulated. The low percentage of blocked calls in PCS networks is exploited by an optimistic scheme for communication within Stage 2. A Stage 2 process optimistically assumes that all handed-off calls to that cell are still allocated in the previous cell. If this assumption holds, no message must be sent. Otherwise, a notification of the contrary must be sent. It may trigger a *rollback* to correct effects of a wrong assumption if the message is *straggler*.

In our mobility model, MHs are classified into groups of *workers*, *wanderers*, *travelers* or *static users*. MHs of a specific group show similar behavior. The call model is specified through intervals of different call ratios during the simulation, to represent congested hours as well as under-loaded hours. The cell topology is specified by a grid embedded in the PCS coverage area, with fine resolution of 50 meters which allows the representation of arbitrary cell sizes and shapes.

4 Performance Analysis

In this section, we present a performance analysis of SWiMNet. First, we investigate the execution time of a (hypothetical) sequential simulation based on SWiMNet, then we examine the execution time and speedup of parallel simulation.

A sequential execution of the simulator requires to produce all precomputed events and store them into a file, and then scan the file and simulate the events. The average elaboration time of one precomputed event is given by $T_{\text{seq}}^{\text{event}} = t_{\text{prod}} + f_{\text{write}} + f_{\text{read}} + t_{\text{sim}}$, where t_{prod} is the average event production time; f_{write} is the average time to write into a file; f_{read} is the average time to read from a file; and t_{sim} is the average event simulation time. Let N_{mh} be the number of mobile hosts, and E be the average number of precomputed events per MH. Thus, the total execution time for a sequential simulation run is:

$$T_{\text{seq}} = N_{\text{mh}} \times E \times (t_{\text{prod}} + f_{\text{write}} + f_{\text{read}} + t_{\text{sim}}) . \quad (1)$$

Let N_1 and N_2 be the number of processors dedicated to Stage 1 and Stage 2 respectively. We assume there is one process per processor, hence the terms process and processor will be used equivalently. The system of processors can be represented as a queueing network as depicted in Fig. 2. In order to compute the

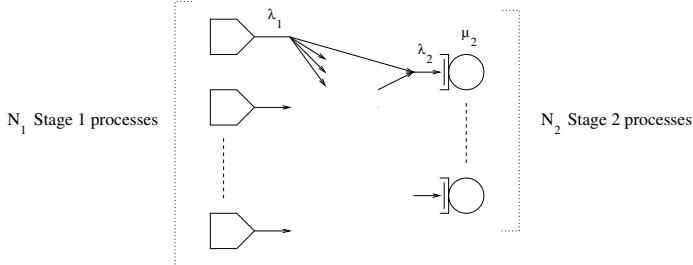


Fig. 2. Queueing model of the parallel simulator

execution time for the parallel simulator, we assume that the precomputation load is evenly distributed among the N_1 processes of Stage 1. This is feasible since MHs of the same group can be evenly distributed among N_1 processes, and MHs of the same group show similar behavior. However, to evenly distribute the workload among the N_2 processes in Stage 2, a careful assessment of loads of different BSs is necessary. In our analysis, we consider both cases, i.e., when the load on Stage 2 is evenly distributed, and when it is unbalanced.

4.1 Estimation of Stage 1 Execution Time

Since a total of $N_{\text{mh}} \times E$ events are generated by the precomputation, each Stage 1 process must generate $\frac{N_{\text{mh}} \times E}{N_1}$ events on the average. Each precomputed event requires an average execution time t_{prod} plus an average send time $t_{1,2}^{\text{send}}$ to Stage 2. Therefore, each Stage 1 processor is a source of jobs at a rate $\lambda_1 = \frac{1}{t_{\text{prod}} + t_{1,2}^{\text{send}}}$, and the execution time of Stage 1 processes is given by

$$T_{\text{par}}^1 = \frac{N_{\text{mh}} \times E}{N_1} \times (t_{\text{prod}} + t_{1,2}^{\text{send}}) . \quad (2)$$

4.2 Estimation of Stage 2 Execution Time

We estimate now the execution time of Stage 2 processes, in a balanced case first and in an unbalanced case later. Because of the assumption of balancement, on the average every Stage 2 process is assigned the same number of precomputed events and the same amount of rollback load. Service times for precomputed events and for rollback messages are different, therefore we model a Stage 2 processor as composed of two servers, S' and S'' (see Fig. 3). The total execution time of Stage 2 is given by the total execution time of two servers.

The amount of jobs from S' to S'' is given by a *rollback initiate probability*, P_{initiate} . It can be estimated as half of the call blocking probability, P_{block} (half, because only call-arrival or move-in events can cause rollbacks while call-termination or move-out events do not), multiplied by the hand-off probability, $P_{\text{h.off}}$, because only handed-off blocked calls generate rollback messages. The average job service time at S' becomes

$$t(S') = t_{\text{sim}} + t_{1,2}^{\text{receive}} + t_{2,2}^{\text{send}} \times \frac{1}{2} \times P_{\text{block}} \times P_{\text{h.off}}$$

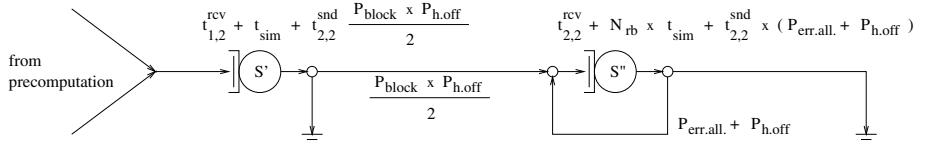


Fig. 3. Queueing model of a Stage 2 process

where t_{sim} is the time for simulating an event; $t_{1,2}^{\text{receive}}$ is the average receive time in Stage 2; and $t_{2,2}^{\text{send}}$ is the average send time in Stage 2.

Rollback jobs have a different service time which is given by

$$t(S'') = t_{2,2}^{\text{receive}} + t_{\text{sim}} \times N_{\text{rb}} + t_{2,2}^{\text{send}} \times (P_{\text{err.all}} + P_{\text{h.off}})$$

where N_{rb} is the average number of jobs involved in a rollback (*coasting-forward* phase), $P_{\text{err.all}}$ is the probability of an erroneous channel allocation (i.e., the probability that a rollback initiates a rollback for another call), and $P_{\text{h.off}}$ is the probability that the same rollback must be forwarded to other processors, in the case that the corresponding erroneous call is still on. The sum of $P_{\text{err.all}}$ and $P_{\text{h.off}}$ is an estimation of the feedback probability at S'' , which we call *rollback cascade probability*, P_{cascade} .

We estimate N_{rb} as the average number of events occurring at one base station while a call is on, assuming that a rollback at one cell involves only one call. Thus, it does not propagate corrections to further calls in the same cell due to changes in channel allocation. Therefore, N_{rb} is given by

$$N_{\text{rb}} = \frac{D \times C \times N_{\text{mh}}}{B} \times 2 \times (1 + E[N_{\text{h.off}}]) \quad (3)$$

where D is the average call duration in hours, C is the average number of calls per MH per hour, B is the number of base stations, and $N_{\text{h.off}}$ is the number of hand-offs per call. The term $2 \times (1 + E[N_{\text{h.off}}])$ counts the average number of events concerning one call. It can be derived from the hand-off probability at a cell, $P_{\text{h.off}}$, by noting that the expected number of hand-offs per call is

$$E[N_{\text{h.off}}] = (1 - P_{\text{h.off}}) \times \sum_{i>0} (i \times P_{\text{h.off}}^i) = \frac{P_{\text{h.off}}}{1 - P_{\text{h.off}}} = \frac{1}{1 - P_{\text{h.off}}} - 1 .$$

In the *balanced* case the arrival rate at a Stage 2 process is $\lambda_2^{\text{bal}} = \lambda_1 \times \frac{N_1}{N_2}$, and the service rate μ_2^{bal} is given by the average execution time per event:

$$\frac{1}{\mu_2^{\text{bal}}} = t(S') + t(S'') \times P_{\text{initiate}} \times \frac{1}{1 - P_{\text{cascade}}} .$$

In the worst *unbalanced* case instead, one Stage 2 process is assigned all the simulation load. Because of this assignment, no rollback is required since Stage 2 is turned into a sequential simulator. The only executing Stage 2 process is modeled as a server with arrival rate $\lambda_2^{\text{unb}} = \lambda_1 \times N_1$, and service rate μ_2^{unb} given by $1/\mu_2^{\text{unb}} = t_{\text{sim}} + t_{1,2}^{\text{receive}}$.

4.3 Speed-Up Estimation

Since job queue lengths must be bounded not to exhaust system resources during the simulation, the condition $\lambda_2^* < \mu_2^*$ must be met, whichever estimation (balanced or unbalanced) of arrival rate λ_2^* and of service rate μ_2^* for Stage 2 are considered. This is a condition of non-saturation for the queueing model. If it holds, the total execution time becomes the time of Stage 1 processes, since Stage 1 never stops precomputing events whereas Stage 2 can spare some inactivity time. Thus, from Equations (1) and (2), the speedup is given by

$$\text{SpeedUp}(N_1, N_2) = \frac{T_{\text{seq}}}{T_{\text{par}}^1} = N_1 \times \frac{t_{\text{prod}} + f_{\text{write}} + f_{\text{read}} + t_{\text{sim}}}{t_{\text{prod}} + t_{1,2}^{\text{send}}} . \quad (4)$$

If we assume that $f_{\text{write}} = f_{\text{read}} = t_{1,2}^{\text{send}} = t_{1,2}^{\text{receive}}$; $N_1 = N_2 = \frac{N}{2}$; and $t_{\text{prod}} = t_{\text{sim}}$, then $\text{SpeedUp}(N_1, N_2) = N$. This shows that our model leads to the maximum speedup achievable. This result is particularly meaningful for the balanced case, since the condition of non-saturation is easier to satisfy.

5 Simulation Study

We simulated a city area serviced by a TDMA based PCS network (the BS map was acquired from a telecommunication company). The coverage area has size 11×11 Km, with 9000 MHs and 54 cells each with 30 channels. Call arrivals are modeled as a Poisson process with 3 calls/hour/MH. Call durations are exponentially distributed with a mean of 120 seconds. We implemented and ran SWiMNet on a cluster of 16 Pentium II (200Mhz) workstations running Debian/Linux, connected via 100 Megabit Ethernet. Process communication is based on MPI.

In addition to measuring the execution time, we consider the following metrics:

- F_{initiate} : an experimental assessment of P_{initiate} .
- F_{cascade} : an experimental assessment of P_{cascade} .
- M_{rb} : the actual average number of events elaborated during a rollback.
- $\rho_{\text{perf.rb}}$: actually performed rollback ratio, i.e., the percentage of straggler Stage 2 messages.

Parameters, estimations and measured values are listed in Table 1. Under “Source” we indicate how each value was obtained. “Benchmark” means a direct measurement of values in the computing environment. Blocking and hand-off probabilities were modeled analytically using an *Erlang-B model* [11].

The estimation of N_{rb} by means of Equation (3) using parameters from Table 1 does not match M_{rb} . This difference is due to $\rho_{\text{perf.rb}}$. In the analytical model the estimation of N_{rb} takes into account all the potential rollbacks, whereas only straggler ones actually have a contribution. The ratio 0.1 brings N_{rb} close to M_{rb} . Also the analytical expression for P_{cascade} does not match F_{cascade} . In fact, even though P_{cascade} depends on $P_{\text{err.all}}$ for which we do not have an analytical expression, $F_{\text{cascade}} = 0.1$ is not larger than $P_{\text{h.off}} = 1/3$ as it should. Again,

Table 1. Parameters assumed in the analytical model

Parameter	Value	Unit	Source
t_{1-2}^{send}	0.001	seconds	Benchmark
t_{1-2}^{receive}	0.001	seconds	Benchmark
t_{2-2}^{send}	0.001	seconds	Benchmark
t_{2-2}^{receive}	0.001	seconds	Benchmark
t_{prod}	0.003	seconds	Benchmark
t_{sim}	0.001	seconds	Benchmark
D	1/30	hours per call	PCS model parameter
C	3	calls per MH per hour	PCS model parameter
N_{mh}	9000	MHs	PCS model parameter
B	54	BSs	PCS model parameter
P_{block}	1/5	probability	Erlang-B model
$P_{\text{h.off}}$	1/3	probability	Erlang-B model
P_{initiate}	1/30	probability	Analytical model
N_{rb}	50	messages	Analytical model
E	7.8	events per MH	Experiment
F_{initiate}	0.036	frequency	Experiment
F_{cascade}	0.010	frequency	Experiment
M_{rb}	4.5	messages	Experiment
$\rho_{\text{perf.rb}}$	0.1	rollback ratio	Experiment

P_{cascade} needs to be scaled down because of $\rho_{\text{perf.rb}}$. An analytical measure which is fairly matched by experimental results is P_{initiate} . According to P_{block} and $P_{\text{h.off}}$, the expected value is 3.33% where the measured value is around 3.6%.

Figure 4 portrays analytical estimations for each stage, and the measured execution time (“experimental results”). The number of processors used in each execution time estimation is half of the corresponding x coordinate. An expected execution time can be derived for any assignment of processors to stages of our simulator, SWiMNet. If N_1 processors are used in Stage 1 and N_2 processors in Stage 2, the analytical execution time is the maximum between the estimation for Stage 1 with $2 \times N_1$ processors, and the estimation for Stage 2 with $2 \times N_2$ processors. However, for reasons of stability, the execution time of Stage 2 shall not be larger than that of Stage 1. The graphs show a close trend between experimental and analytical data. This indicates a good predictability of the simulator’s performance. The linear actual speedup shows a good scalability.

6 Conclusions

We have presented a performance model of a parallel simulator of PCS networks, called SWiMNet. This simulator is meant to help PCS network designers, by speeding up the design and testing of realistic PCS network models. SWiMNet exploits event precomputation which is possible due to independence of PCS model components. Process coordination in SWiMNet is a hybrid of conservative and optimistic techniques.

The analytical model is based on queueing models and leads to a linear speedup on the number of processors used. The performance model has been compared with actual experiments conducted on a real PCS network model. The measured performances closely match the analytical estimation.

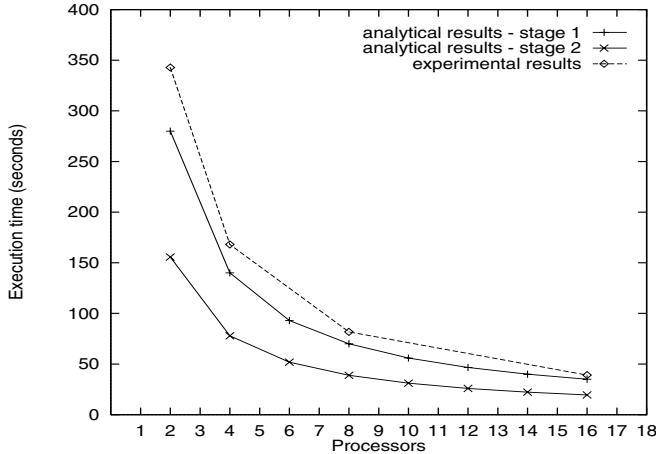


Fig. 4. Analytical vs. experimental execution time

References

1. A. Boukerche, S.K. Das, A. Fabbri, O. Yildiz, Exploiting Model Independence for Parallel PCS Network Simulation, *Proceedings of the 13th Workshop on Parallel And Distributed Simulation* (PADS'99), 166-173.
2. C. Carothers, R. Fujimoto, Y.-B. Lin, P. England, Distributed Simulation of Large-Scale PCS Networks, *Proceedings of the 2nd Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS'94), 2-6.
3. K.M. Chandy, J. Misra, Distributed Simulation: A Case Study in Design and Verification of Distributed Programs, *IEEE Transactions on Software Engineering*, Vol.SE-5 (September 1979), 440-452.
4. R.M. Fujimoto, Parallel Discrete Event Simulation, *Communications of the ACM*, Vol.33, No.10 (October 1990), 30-53.
5. D.R. Jefferson, Virtual Time, *ACM Transactions on Programming Languages and Systems*, Vol.7, No.3 (July 1985), 404-425.
6. C.Y. Lee William, Mobile Cellular Telecommunications: Analog and Digital Systems, (McGraw-Hill Inc., 1989).
7. M. Liljenstam, R. Ayani, A Model for Parallel Simulation of Mobile Telecommunication Systems, *Proceedings of the 4th Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS'96).
8. Y.-B. Lin, P. Fishwick, Asynchronous Parallel Discrete Event Simulation, *IEEE Transactions on Systems and Cybernetics*, Vol.26, No.4 (July 1996), 397-412.
9. R.A. Meyer, R.L. Bagrodia, Improving Lookahead in Parallel Wireless Network Simulation, *Proceedings of the 6th Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS'98), 262-267.
10. J. Panchal, O. Kelly, J. Lai, N. Mandayam, A. Ogielski, R. Yates, WIPPET, A Virtual Testbed for Parallel Simulations of Wireless Networks, *Proceedings of the 12th Workshop on Parallel And Distributed Simulation* (PADS'98), 162-169.
11. T.D. Rappaport, Wireless Communications: Principles and Practice (Prentice Hall, 1996).
12. X. Zeng, R. Bagrodia. GloMoSim: A Library for the Parallel Simulation of Large Wireless Networks, *Proceedings of the 12th Workshop on Parallel And Distributed Simulation* (PADS'98), 154-161.

Ultimate Parallel List Ranking ?

Jop F. Sibeyn

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany.
URL: <http://www.mpi-sb.mpg.de/~jopsi/>

Abstract. Two improved list-ranking algorithms are presented. The “peeling-off” algorithm leads to an optimal PRAM algorithm, but was designed with application on a real parallel machine in mind. It is simpler than earlier algorithms, and in a range of problem sizes, where previously several algorithms were required for the best performance, now this single algorithm suffices. If the problem size is much larger than the number of available processors, then the “sparse-ruling-sets” algorithm is even better. In previous versions this algorithm had very restricted practical application because of the large number of communication rounds it was performing. This weakness is overcome by adding two new ideas, each of which reduces the number of communication rounds by a factor of two.

1 Introduction

Problem Definition. The input is a set of lists or trees of total length N . Every node has a pointer to a successor, stored in a field $succ$. A final node j can be recognized by a distinguished value of $succ(j)$. The output consists of two arrays: for every $0 \leq j < N$, the master of j , $mast[j]$, should give the index of the final node of the list or tree to which j belongs, and $dist[j]$ should give the number of links between j and $mast[j]$. The number of PUs is denoted P , and every PU holds exactly $k = N/P$ nodes of the lists: $PU_i, 0 \leq i < P$, holds the nodes with indices $k \cdot i + j$, for all $0 \leq j < k$.

Cost Model. We express the quality of our parallel algorithms by giving their *routing volume*, the number of integers sent by a PU, and the number of all-to-all routing operations (informally we call this the number of communication rounds). Our cost measure is a simplification of BSP or BSP* [8,3,1]. In [7] it was shown to be accurate.

Earlier Work. The parallel list-ranking problem has extensively been investigated. The most relevant recent papers are [5,2,4].

2 Independent-Set Removal

In the independent-set-removal algorithm, reductions are repeated until the problem size has become sufficiently small to terminate with some other algorithm. Then the excluded nodes are reinserted in reverse order. At all times, there is a list of active nodes. Initially, all non-final nodes p are active and set $mast(p) = succ(p)$ and $dist(p) = 1$. In Phase t of the reduction we perform

Algorithm REDUCTION(t)

1. Each active node chooses independently a 0 or a 1 with probability 1/2. Each node p that has chosen a 1 sends a packet to $mast(p)$.

2. If a node p which selected a 0 receives a packet, then it is removed from the list of active nodes and added to the list of nodes excluded during Phase t . It sends $mast(p)$ and $dist(p)$ back to the sending node. Otherwise it sends back the number -1 .

3. If an active node p receives -1 , then it does nothing. Otherwise it uses the received data to update $mast(p)$ and $dist(p)$.

Every phase reduces the problem size to about $3/4$. The reinsertion is even simpler.

Algorithm REINSERTION(t)

1. Each node that was excluded during Phase t sends a packet to its master.
2. Each node p that received a packet sends back $mast(p)$ and $dist(p)$.
3. Each node p that was excluded during Phase t uses the received data to update $mast(p)$ and $dist(p)$.

$\log_2 k$	$P = 4$	$P = 16$	$P = 64$
12	0.09	0.06	0.04
14	0.18	0.13	0.09
16	0.22	0.19	0.16
18	0.22	0.20	0.17
20	0.21	0.20	0.18

Table 1. Efficiencies of independent-set removal, running on an Intel Paragon for various P and k . In all cases we performed ten reduction phases. By efficiency we mean $speed-up/P = T_{\text{seq}}/(P \cdot T_{\text{par}})$. As a basis for the computation of our efficiencies, we assumed that $T_{\text{seq}} = 3.9 \cdot 10^{-6} \cdot N$, for all N : running on one node of the Paragon, the simple sequential algorithm requires 3.9s for solving a problem with $N = 10^6$.

Theorem 1 A parallel implementation of the independent-set-removal algorithm has routing volume $(8 + o(1)) \cdot k$, with high probability. One level of REDUCTION and INSERTION requires 4 all-to-all routings.

3 Peeling-Off Algorithm

Basic Idea. The basic idea of our first algorithm is to split the nodes into two sets: \mathcal{S}_0 and \mathcal{S}_1 . The set of all nodes is denoted \mathcal{N} . Then we perform

Algorithm PEELING_OFF($\mathcal{S}_0, \mathcal{S}_1$)

1. AUTOCLEAN(\mathcal{S}_1);
2. ALTROCLEAN(\mathcal{S}_0);
3. SOME_RANK(\mathcal{S}_0);
4. ALTROCLEAN(\mathcal{S}_1).

Here SOME_RANK may be any ranking algorithm. By AUTOCLEAN(\mathcal{S}_j) we mean:

All nodes in \mathcal{S}_j follow the links running through nodes in \mathcal{S}_j until a link out-off \mathcal{S}_j is found or a final node is reached. Then they update $mast$ and $dist$.

By ALTROCLEAN(\mathcal{S}_j) we mean:

All nodes in \mathcal{S}_j that have not reached a final node and whose master is not an element of \mathcal{S}_j , ask their master for its $mast$ and $dist$ fields. Then they update their $mast$ and $dist$ fields with the received values.

Lemma 1 If $\text{mast}(j) = \text{succ}(j)$, $\text{dist}(j) = 1$, then PEELING_OFF correctly computes the values of mast and dist for all nodes.

PRAMs. On a PRAM one should first perform some randomization: every node is placed in a randomly chosen bucket of size $N / \log N$. With P PUs this can be done in $\mathcal{O}(N/P + \log N)$ time. The buckets are numbered from 0 through $\log N - 1$, and the set of nodes in Bucket j is denoted Buc_j . Then we perform $\log \log N$ rounds of PEELING_OFF in which the problem size is halved each time. In Round t , $1 \leq t \leq \log \log N$, we take

$$\begin{aligned}\mathcal{S}_0(t) &= \bigcup_{j=0}^{\log N/2^t-1} Buc_j, \\ \mathcal{S}_1(t) &= \bigcup_{j=\log N/2^t}^{\log N/2^{t-1}-1} Buc_j.\end{aligned}$$

Finally pointer jumping is performed on $\mathcal{S}_0(\log \log N) = Buc_0$.

Theorem 2 On a PRAM with P PUs, PRAM_RANK ranks a set of list with N nodes in $\mathcal{O}(N/P + \log N)$ time, with high probability. The same algorithm solves tree routing in $\mathcal{O}(N/P + \log N)$ expected time.

Distributed Memory Machines. On a distributed memory machine PU_i , holds the $k = N/P$ nodes with indices $i + j \cdot P$, for all $0 \leq j < k$. Each PU has a buffer for every PU, in which it writes questions and answers. In any step, all questions or answers are generated, then the all-to-all routing is performed and so on. This is the standard way of running algorithms under the BSP paradigm. To optimize the algorithm, we use one-by-one cleaning from [7] instead of pointer-jumping:

Lemma 2 [7] For ranking a set of lists with a total of $k \cdot P$ nodes on a parallel computer with P PUs, one-by-one cleaning requires $3 \cdot P - 4$ start-ups, and has routing volume $6 \cdot \ln P \cdot k$.

$\log_2 k$	$P = 4$	$P = 16$	$P = 64$
10	0.06	0.03	0.01
12	0.19	0.07	0.03
14	0.36	0.10	0.09
16	0.47	0.29	0.18
18	0.44	0.35	0.23
20	0.41	0.35	0.26

Table 2. Efficiencies of the parallel algorithm running on an Intel Paragon for various P and k .

It is important to optimize the choice of the number of reduction rounds d and the reduction factors: the number f_t , $1 \leq t \leq d$, given by $f_t = \#\mathcal{S}_1(t)/\#\mathcal{S}_1(t-1)$, where $\#\mathcal{S}_1(0) = N$. These must be tuned to obtain a good trade-off between the number of all-to-all routings and the routing volume. A handy choice is

$$f_t(d) = (1 + d - t)/(2 + d - t). \quad (1)$$

With these f_t , we get a simple expression: $\#\mathcal{S}_0(t) = (d + 1 - t)/(d + 1) \cdot N$.

Theorem 3 When d reduction phases are performed with reduction factors as in (1), the routing volume is less than $(6 + (3 \cdot \ln d + 6 \cdot \ln P)/(d + 1)) \cdot k$, with high probability. For each reduction phase, the algorithm requires $6 + 2 \cdot \lceil \log \log N \rceil$ all-to-all routings.

4 Sparse-Ruling-Set Algorithm

Earlier versions of the sparse-ruling-set algorithm were given in [5,6,4]. Its basic idea allows for a highly efficient reduction of the problem size by a large factor: during the whole algorithm, most nodes are addressed only twice. This is not more than in the sequential algorithm, and therefore one would hope to obtain very high speed-ups. Unfortunately, on parallel computers with a distributed memory, good performance is achieved only when N/P^2 is very large, because the number of communication rounds is too high. Here we introduce two important new ideas that allow to reduce the number of communication rounds considerably.

Constant Number of Active Waves. Consider the following algorithm:

Algorithm SPARSE-RULING-SET(N, S)

1. Select S nodes randomly and uniformly from among the non-final nodes as *rulers*.
2. Each ruler p initiates a *wave*: p prepares a packet containing its index and $dist(p)$ to be sent to $succ(p)$.
3. Mark all final nodes as rulers.
4. **for** $round = 0$ **to** $N/S - 1$ **do**
 - a. Send all packets.
 - b. Each node p that receives a packet marks the contained data.
 - c. **if** p is a non-ruler **then**
 p prepares a new packet for $succ(p)$, adding $dist(p)$ to the second field in the packet.
 - else**
A new ruler p' is selected from among the unreached non-rulers. p' initiates a new wave as was done in Step 2.

In every round, S nodes are reached that were not reached before. Thus, all nodes have been reached after N/S rounds. All non-initial rulers are reached by waves from their predecessors, and thus a sublist with links consisting of all rulers can be constructed without further communication. The rest of the algorithm is the same as in independent-set removal: after some more reduction rounds, pointer-jumping or one-by-one cleaning is performed, and finally the excluded nodes are reinserted.

Theorem 4 *On an interconnection network, an application of SPARSE-RULING-SET causes a routing-volume of $3 \cdot k$. The expected number of rulers selected by SPARSE-RULING-SET(N, S) equals $S \cdot H_{N/S} = S \cdot \sum_{i=1}^{N/S} 1/i$.*

Corollary 1 *Asymptotically the presented sparse-ruling-set algorithm is twice as effective as earlier versions. Here the effectiveness is measured by the number of routing steps for achieving a given reduction of the problem size.*

Interlacing Computation and Communication. In SPARSE-RULING-SET, an all-to-all routing is performed in Step 4.a. Instead of this, the all-to-all routing might also be decomposed into $P - 1$ permutations. For example, PU i might send in Step j , $1 \leq j < P$, to PU $(i + j) \bmod P$. After routing each such a permutation, the received

data can be processed immediately as in Step 4.b and 4.c, before routing the following packet. If the successor of a newly reached node is residing in the same PU, then a shortcut can be made without waiting. In the original algorithm, a wave is progressing exactly one step after every full communication round, now it is progressing twice as fast.

Theorem 5 *If routing rounds are decomposed in $P - 1$ permutation routings interlaced with data processing, then all nodes are reached in $N/(2 \cdot S) + 1$ rounds, with high probability.*

Corollary 2 *Asymptotically, interlacing computation and communication makes SPARSE-RULING-SET twice as effective.*

Experiments. A sequential simulation of SPARSE-RULING-SET with interlaced computation and communication has been programmed in C. It is available at <http://www.mpi-sb.mpg.de/~jopsi/dprog/prog.html>. The simplicity of the algorithm leads to a very short and simple code. Another great advantage is that both new ideas help to reduce buffer sizes. In addition to the $3 \cdot N$ integers for storing the *succ*, *mast* and *dist* fields, our program requires less than $N/2$ additional storage.

N	S	Rounds	N'
4194304	524288	6	1612992
4194304	131072	18	584288
4194304	32768	66	191216

Table 3. Reductions of the problem sizes by SPARSE-RULING-SET with interlaced computation and communication. The columns give N , S , the number of communication rounds, and the resulting problem size.

References

1. Bäumker, A. W. Dittrich, F. Meyer auf der Heide, ‘Truly Efficient Parallel Algorithms: c-Optimal Multisearch for an Extension of the BSP-Model,’ *Proc. 3rd European Symposium on Algorithms*, LNCS 979, Springer-Verlag, pp. 17–30, 1995.
2. Hsu, T.-s, V. Ramachandran, ‘Efficient Massively Parallel Implementation of some Combinatorial Algorithms,’ *Theoretical Computer Science*, 162(2), pp. 297–322, 1996.
3. McColl, W.F., ‘Universal Computing,’ *Proc. 2nd Euro-Par Conference*, LNCS 1123, pp. 25–36, Springer-Verlag, 1996.
4. Ranade, A., ‘A Simple Optimal List Ranking Algorithm,’ *Proc. of 5th High Performance Computing*, Tata McGraw-Hill Publishing Company, 1998.
5. Reid-Miller, M., ‘List Ranking and List Scan on the Cray C-90,’ *Journal of Computer and System Sciences*, 53(3), pp. 344–356, 1996.
6. Sibeyn, J.F., ‘List Ranking on Interconnection Networks,’ *Proc. 2nd Euro-Par Conference*, LNCS 1123, pp. 799–808, Springer-Verlag, 1996.
7. Sibeyn, J.F., ‘Better Trade-offs for Parallel List Ranking,’ *Proc. 9th Symposium on Parallel Algorithms and Architectures*, pp. 221–230, ACM, 1997.
8. Valiant, L.G., ‘A Bridging Model for Parallel Computation,’ *Communications of the ACM*, 33(8), pp. 103–111, 1990.

A Parallel 3-D Capacitance Extraction Program*

Yanhong Yuan and Prithviraj Banerjee

Department of Electrical and Computer Engineering
Northwestern University, Evanston, IL 60208, USA
`{yuanyh, banerjee}@ece.nwu.edu`

Abstract. This paper presents a scalable data-partitioned parallel algorithm for accurate 3-D capacitance extraction using the Boundary Element Method(BEM). We address the task decomposition and workload balancing issues when system matrix irregularity presents due to applying the adaptive Gaussian quadrature. Experimental results are reported on an IBM SP2, an IBM J40 and a network of HP workstations.

1 Introduction

Accurate estimation of the capacitances in complicated three dimensional(3-D) interconnects is getting increasingly important for determining the final circuit speeds or functionality in the ultra deep sub-micron design(UDSM). Although 3-D simulation tools[2, 3, 7] can help designers find signal integrity problems, even the fastest of these tools running on a scientific workstation are too slow to allow a designer to quickly investigate a variety of conductor layouts. Several parallel extraction algorithms have been proposed to reduce the running time of the extraction tools[5, 9]. Due to the inherent heavy communication requirement or the difficulty to find an efficient task decomposition, the parallel efficiency was restricted, and a speedup of 5 on 8 processors was reported.

This paper examines the parallelization of another capacitance extraction method, the direct Boundary Element Method (BEM)[1, 2]. To sparsify the associated dense matrix and speed up its construction, the adaptive quadrature scheme[10] is used for the potential integral evaluation. However, due to the use of the adaptive scheme, each entry of the system matrix will involve a different amount of computation and the matrix configuration will exhibit various level of *irregularity*. Therefore, a simple block or even cyclic mapping can not ensure a good load balancing. In this paper, we address the task decomposition and workload balancing of the parallel algorithm in such case. The parallel algorithm has been developed using the Message Passing Interface(MPI). Very good speedups have been obtained on a variety of parallel platforms.

* This research was supported in part by the Advanced Research Projects Agency under contract DAA-H04-94-G0273 administered by the Army Research Office.

2 Background

Problem Formulation To solve the capacitance problem using the direct BEM method[1], the following integral equation should be efficiently solved:

$$cu + \int_{\Gamma} q^* u d\Gamma = \int_{\Gamma} u^* q d\Gamma, \quad (1)$$

where u is the potential at a point, and c_s is a constant depending on the geometry of the boundary Γ . In a 3-D situation, the fundamental solution u^* is $\frac{1}{4\pi r}$. q is the normal electric field. A linear system of equations can be obtained by discretizing the above integral equation:

$$Ax = f, \quad (2)$$

where the 2-D surface integrations with the kernel $1/r^l, l \geq 1$ should be evaluated. The normal electric field q on the Dirichlet boundary can be directly obtained from the solution of (2), and the capacitance can be computed as: $C = \int_{\Gamma_u} \epsilon_d q d\Gamma$, where ϵ_d is the permittivity. Iterative algorithms like GCR[3] or GMRES[8] is usually used to solve the linear system.

The Automatic Gaussian Quadrature Gaussian quadratures are often employed in the integral evaluations. A set of formulas were developed in [10] which offer users a freedom when using Gaussian quadrature in their application, that is, to *automatically choose the appropriate integration order according to a user specified error bound ϵ* . The formulas suggest that when the observation point is relatively far from the source region, lower order quadratures can be used for a given error ϵ . It was shown that an order of magnitude speedup can be obtained using the adaptive quadrature, compared with the fix order quadrature.

3 Parallel Implementation

3.1 Task Decomposition and Mapping

One method to distribute the work is to allocate equal numbers of elements to each processor. Then each processor generates only those matrix entries that have interactions with the elements it owns, and further computes the partial matrix-vector product. However, due to the application of adaptive quadrature, each entry of the matrix may require a different computation effort. A simple block or even a cyclic mapping may result in unbalanced load(Figure 1).

The idea is to find a permutation matrix P such that $B = P^{-1}AP$, and all strong interactions should be close to the diagonal in the matrix. In this case, a cyclic mapping can result in a well balanced workload. For 3-D problems, there is generally no such permutations P that can completely accomplish this. However, the heuristics based on the recursive subdivision[7] yields excellent results and require only $O(n \log n)$ time on one processor. It is done by equally dividing a point set S into two disjoint sets S_1 and S_2 , and recursively order S_1 and S_2 and concatenate the results to obtain the ordering for S .

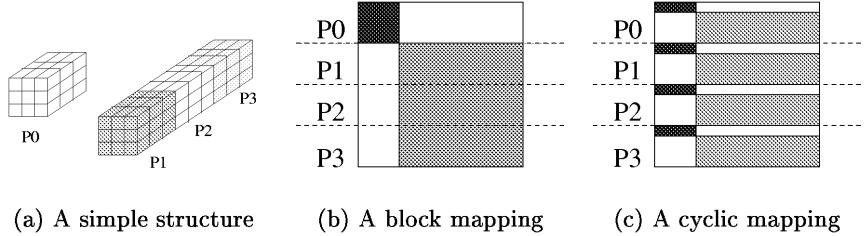


Fig. 1. A block mapping(b) will result in unbalanced workload. A cyclic mapping(c) may result in balanced workload, however, this is not guaranteed for all cases.

3.2 The Parallel Algorithms

An overview of the approach to parallelization is shown schematically in Figure 2. To handle complicated interconnect structures, a *pre-processing* algorithm is designed to perform the conductor surface partition and evenly distribute the workload among all processors.

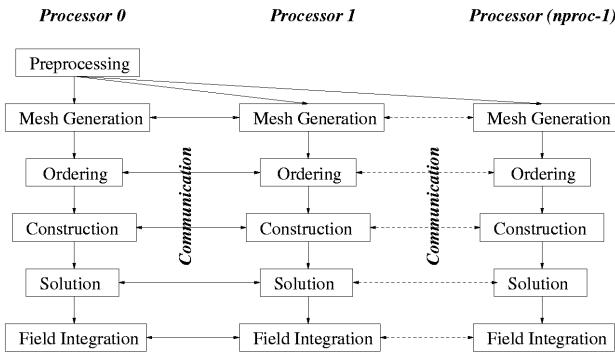
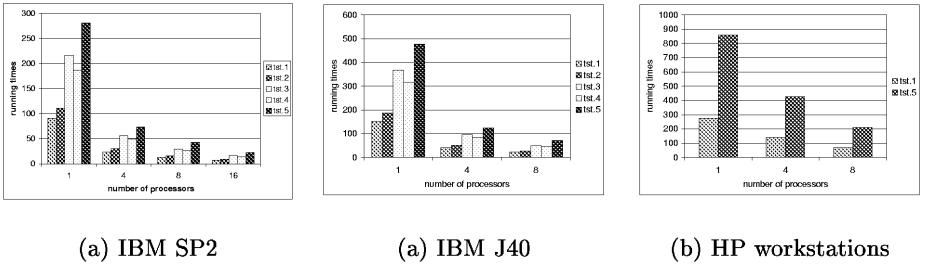


Fig. 2. An overview of the parallel approach.

Parallel Ordering To recursively compute the ordering of elements, we configure the computation as a binary tree. Processor 0 will divide the original set S into two sets S_1 and S_2 . Then S_2 will be sent to processor 1 for recursive ordering and processor 0 will continue working on set S_1 . Generally, at level l , after processor $p, p \in [0, 2^l)$ sub-divides a set into two sets, one of them will be sent to processor $(p + 2^l)$ for further recursive partition and ordering.

Parallel System Construction After the ordering, the elements will be distributed among processors using the cyclic mapping. A linear system will be

**Fig. 3.** Running times in seconds.

constructed by discretizing the integration equation(1) using the mesh generated in the previous step. Because of the ordering, the cyclic mapping will ensure a good load balancing in the matrix construction.

Parallel System Solution As mentioned above, the rows of the matrix have been evenly distributed among processors cyclically. Thus, each processor just performs the matrix-vector product on the rows it owns, and the residual and solution vectors of each iteration will be broadcasted to each processor to perform the computation of the next iteration.

Typical message passing is performed by MPI *all-gather*, *all-reduce* and *broadcast* operations. The total running time for the parallel computation is $O(sn^2/P) + O(s \log P)$, where n is the total number of elements, P is the number of processors used and s is the number of iterations for system convolution.

4 Experimental Results

A set of structures[4, 6, 10] were employed to test the performance of the parallel program. The running times are summarized in Figure 3. An average speedup of 12 is reported on a distributed memory IBM SP2 with 16 processors. On an IBM J40 shared memory multiprocessor, the average speedup is about 6.9 on 8 processors. A speedup of about 4 is obtained on a network of 8 HP workstations.

The ordering has improved the load balancing under cyclic mapping and therefore reduced the running time. The running times on 4 processors of an IBM SP2 are listed in Table 1 .

Two larger 3-D interconnect structures[6] could not be fit in one or four processors as the program could allocate only about 100MB of memory per processor. Table 2 gives the running times and the estimated speedups.

5 Conclusions

We presented a scalable data-partitioned parallel algorithm for 3-D capacitance extraction using the direct BEM model. We addressed the task decomposition

Table 1. Running times in seconds on 4 processors of an IBM SP2.

structure	tst.1	tst.2	tst.3	tst.4	tst.5
no ordering	25.6	32.3	70.2	62.6	80.7
with ordering	23.7	30.5	56.3	49.2	73.0

Table 2. Times in seconds and estimated speedups for large structures on IBM SP2.

processors	1	4	8	12	16
large1	NA(1.0)	602(3.6)	309(7.0)	202(10.5)	163(13.0)
large2	NA(1.0)	NA(NA)	471(7.0)	329(10.3)	266(12.8)

and load balancing under the irregularity. Experimental results showed that the parallel algorithm is efficient and can achieve very good speedups on a variety of parallel platforms.

References

1. Brebbia, C.A.: *The Boundary Element Method for Engineers*. Pentech Press, London (1978)
2. Fukuda, S., Shigyo, N., Kato, K., Nakamura, S.: A ULSI 2-D Capacitance Simulator for Complex Structures Based on Actual Processes. *IEEE Trans. CAD.* **9** (1990) 39–47
3. Nabors, K., White, J.: FASTCAP: A Multipole-Accelerated 3-D Capacitance Extraction Program. *IEEE Trans. CAD.* (1991) 1447–1459
4. Nabors, K., White, J.: Multipole-Accelerated Capacitance Extraction Algorithms for 3-D Structures with Multiple Dielectrics. *IEEE Trans. on CAS.* **11** (1992) 946–954
5. Wang, Z., Yuan, Y., Wu, Q.: A Parallel Multipole Accelerated 3-D Capacitance Simulator Based on an Improved Model. *IEEE Trans. CAD.* **12** (1996) 1441–1450
6. Yuan, Y., Banerjee, P.: Parallel Extraction of 3-D Capacitance and Resistance. Technical Report. ECE Dept. Northwestern University (1998)
7. Kapur, S., Long, D.E.: *IES³*: A Fast Integral Equation Solver for Efficient 3-Dimensional Extraction. *Proc. of ICCAD.* (1997) 448–455
8. Saad, Y., Shultz, M.H.: GMRES: A Generalized Minimal Residual Algorithm for solving non-symmetric linear systems. *SIAM J. of Sci. and Stat. Comput.* **7** (1986) 856–869
9. Aluru, N.R., Nadkarni, V.B., White, J.: A Parallel Precorrected FFT Based Capacitance Extraction Program for Signal Integrity Analysis. *Proc. of DAC.* (1996)
10. Yuan, Y., Banerjee, P.: Fast Potentail Integrals for 3-D Capacitance Extraction. *Proc. of TAU99.* (1999) 53–57

Parallel Algorithms for Queries with Aggregate Functions in the Presence of Data Skew

Y. Jiang, K.H. Liu, and C.H.C. Leung

School of Communication and Informatics
Victoria University of Technology
Melbourne, AUSTRALIA
jiang@matilda.vu.edu.au
clement@matilda.vu.edu.au

Abstract. Queries with aggregate functions often involve more than one relation and thus join is needed. In this paper, the effect of the sequence of aggregation and join operations is identified in a parallel processing environment. Three parallel methods of processing aggregate function queries are developed which mainly differ in the selection of partitioning attribute. Their cost models that incorporate the effect of data skew are provided.

1. Introduction

With the increasing complexity of database applications and the rapid improvement in multiprocessor technology, parallel query processing has emerged as an efficient way to improve database performance. Parallel processing of major relational operations, such as selection, projection and join, has been studied extensively in recent years. By contrast, parallel aggregation receives much less attention although it is critical to the performance of database applications such as OLAP, quality control, and in particular data warehouse [2]. As a repository of integrated decision making information, data warehouses require efficient execution of aggregate functions. Large historical tables need to be joined and aggregated each other; consequently, the effective optimisation of aggregate functions has the potential for huge performance gains.

This paper presents three parallel processing methods for queries involving aggregations and joins, namely, *join-partition method (JPM)*, *aggregation-partition method (APM)* and *hybrid-partition method (HPM)*. *JPM* and *APM* mainly differ in the selection of partitioning attribute for distributing workload over processors and *HPM* is an adaptive method based on *APM* and *JPM* with a logical hybrid architecture. The presented methods are different from previous work in that both joins and aggregate conditions are considered in the optimisation [1] [3] [5] [7]. Moreover, our methods take into account the problem of data skew since the skewed load distribution may affect query execution time significantly.

2. Parallelising Aggregate Functions

For simplicity of description and without loss of generality, we consider queries that involve only one aggregation function and a single join. An example query is given below which clusters the part shipment by their city locations and selects the cities with average quantity of shipment between 500 and 1000.

```
Query 1: SELECT parts.city, AVG(qty)
          FROM parts, shipment
          WHERE parts.pno = shipment.pno
          GROUP BY parts.city
          HAVING AVG(qty)>500 AND AVG(qty)<1000;
```

For parallel query processing, we assume a parallel database architecture that consists of a host and a set of work processors. The host accepts queries from users and distributes each query with required base relations to the processors for execution. The processors perform the query in parallel with possibly intermediate data transmission between each other through the network, and finally send the result of the query to the host. An aggregation query is carried out in three phases: (1) *data partitioning* in which the operand relations of the query are partitioned and the fragments are distributed to all processors; (2) *parallel processing* in which the query is executed in parallel by the processors and the intermediate results are produced; and (3) *data consolidation* in which the final result of the query is obtained by consolidating the intermediate results from the processors.

3. Parallel Processing Methods

Two important decisions need to be made in the parallelisation procedure. One is the selection of partition attribute, that either join attribute or group-by attribute may be considered. The other is the execution sequence of the aggregation and join operations. We present three parallel processing methods that are different from each other in these two decisions.

3.1 Join Partitioning Method (*JPM*)

The *JPM* can be stated as follows. First, the relations R and S are partitioned into N fragments in terms of join attribute, i.e. the tuples with the same join attribute values in the two relations fall into a pair of fragments. Each pair of the fragments will be sent to one processor for execution. Upon receipt of the fragments, in parallel, the processors perform the join operation followed by a local aggregation operation on the fragments allocated. After that, a global aggregation operation is carried out by redistributing the local aggregation results across the processors such that the result tuples with identical values of group-by attribute are allocated to the same processors.

Then, each processor performs a N -way merge with the intermediate aggregation results, followed by doing a restriction operation for the *Having* clause if it exists. Finally, the host consolidates the partial results from the processors by a simple union operation, producing the query result.

Given the notation below,

r, s	the number of tuples in base relations R and S
r_i, s_i	the number of tuples of fragments of R and S at processor i
$Sel(i)$	join selectivity factor for fragment i
$Agg(i)$	aggregation factor for fragment i
θ	reduction factor after performing <i>Having</i> clause
T_{comm}	the average data transmission time for each message
T_{join}	the average join time for each tuple
T_{agg}	the average aggregation time for each tuple

the execution time for the *JPM* method can be expressed as

$$\begin{aligned} & T_{comm} \times (\max(r_i + s_i)) + T_{join} \times (\max(r_i \log s_i)) + T_{agg} \times (\max(r_i \times s_i \times Sel(i))) \\ & + (T_{comm} + T_{agg}) \times (\max(r_i \times s_i \times Sel(i) \times Agg(i))) \\ & + T_{comm} \times (\max(r_i \times s_i \times Sel(i) \times Agg(i) \times \theta)) \end{aligned} \quad (1)$$

3.2 Aggregation Partitioning Method (*APM*)

In the *APM* method, the relation with group-by attribute, say R , is partitioned into N fragments in terms of the group-by attribute, i.e. the tuples with identical attribute values will be allocated to the same processor. The other relation S needs to be broadcast to all processors in order to perform the binary join. After data distribution, each processor first conducts the joining of one fragment of R with the entire relation S , followed by the group-by operation and *having* restriction if it exists. Since the relation R is partitioned on group-by attribute, the final aggregation result can be simply obtained by a union of the local aggregation results from the processors, i.e. the step of merging local results used in *JPM* method is not required. Consequently, the cost of the *APM* method is given by

$$\begin{aligned} & T_{comm} \times (\max(r_j + s)) + T_{join} \times (\max(r_j \log s)) + T_{agg} \times (\max(r_j \times s \times Sel(j))) \\ & + T_{comm} \times (\max(r_j \times s \times Sel(j) \times Agg(j) \times \theta)) \end{aligned} \quad (2)$$

3.3 Hybrid Partitioning Method (*HPM*)

The *HPM* method is a combination of the *JPM* and *APM* methods. In the *HPM*, the processors are divided into m clusters each of which has N/m processors. The data partitioning phase is carried out in two steps. First, the relation with group-by attributes is partitioned into processor clusters in the same way of the *APM*; i.e. partitioning on the group-by attribute and the other relation is broadcast to each of the clusters. Second, within each cluster, the fragments of the first relation and the entire second relation is further partitioned by the join attributes as the *JPM* does. Depending on the parameters such as the relation's cardinalities and the skew factors, a proper value of m can be chosen such that the minimum query execution time is achieved. More precisely, the *HPM* method is described below:

1. Partition the relation R on group-by attribute to m clusters, denoted by r_i . Within each cluster, further partition the fragment r_i and the other relation S on join attribute to $n=N/m$ processors, denoted by r_{ij} and s_j .
2. Carry out the join and perform local aggregation at each processor.
3. Redistribute the local aggregation results to the processors within each cluster by partitioning the results on the group-by attribute.
4. Merge the local aggregation results within each cluster.
5. Perform the *Having* predicate in each cluster.
6. Transfer the results from the clusters to the host, followed by result consolidation.

The total execution time of the *HPM* is the sum of the times of the above steps, i.e.,

$$\begin{aligned} & T_{comm} \times (\max(r_{ij} + s_j)) + T_{join} \times (\max(r_{ij} \log s_j)) + T_{agg} \times (\max(r_{ij} \times s_j \times Sel(j))) \\ & + (T_{comm} + 2T_{agg}) \times (\max(r_{ij} \times s_j \times Sel(j) \times Agg(j))) \\ & + T_{comm} \times (\max(r_{ij} \times s_j \times Sel(j) \times Agg(j) \times \theta)) \end{aligned} \quad (3)$$

The execution time for each of the components in the above equation varies with the degree of skewness, and could be far from the average [6]. Therefore, we introduce three skew factors α_n , α_m and β_n , where m is the number of processor clusters and n is the number of processors in each cluster. α_m and α_n describe the data partitioning skew among and within the clusters; while β_n represents the data processing skew within the clusters [4]. Assuming that the partitions of both operands R and S follow Zipf distribution, we may approximate $\alpha_n = \frac{1}{\gamma + \ln n}$ and $\alpha_m = \frac{1}{\gamma + \ln m}$, where $\gamma = 0.57721$ known as the Euler constant.

The skew factor β_n is affected by the data partitioning skew factors in both relations R and S since the join/aggregation results rely on the operand fragments. The actual value is therefore usually difficult to estimate accurately. We assume here

$\beta_n = (\alpha_n^2 + 1) / 2$. Given few simplifications that $J_i = r_i \times s_i \times Sel(i) = J$, $Agg(j) = \frac{1}{m} Agg$, $T_{join} = T_{agg} = T_{proc}$, and z being message size, the cost of HPM can be re-written as

$$\begin{aligned} & T_{comm} \left[\alpha_n \times \alpha_m \times r + \alpha_n \times s + \frac{(1 + \theta)}{\beta_n} \times J \times \frac{Agg}{m} \right] / z \\ & + T_{proc} \left[\alpha_n \times \alpha_m \times r \times \log(\alpha_n \times s) + \frac{J}{\beta_n} \times \left(1 + 2 \times \frac{Agg}{m} \right) \right] \end{aligned} \quad (4)$$

4. Conclusion

Traditionally, join operation is processed before aggregation operation and relations are partitioned on join attribute. We demonstrated in this paper that group-by attribute may also be chosen as the partition attribute. Three parallel methods are proposed which differ in the way of distributing operand relations, i.e. based on either the join attribute, the group-by attribute, or a combination of both. Consequently, different query execution costs are incurred. The problem of data skew has also been taken into account in the proposed methods as it may adversely affect the performance advantage of parallel processing.

References

1. Bultzingsloewen G., "Translating and optimizing SQL queries having aggregate", in *Proceedings of the 13th International Conference on Very Large Data Bases*, 1987
2. Datta A. and Moon B., "A case for parallelism in data warehousing and OLAP", in *Proceedings of 9th Intern. Workshop on Database and Expert Systems Applications*, 1998
3. Gendrano J.A., Huang B.C. and Rodrigue J.M., "Parallel algorithms for computing temporal aggregates", *Proceedings of 15th Intern. Conference on Data Engineering*, 1999
4. Liu K. H., Leung C. H. C., and Y. Jiang, "Analysis and taxonomy of skew in parallel database systems", in *Proceedings of the International Symposium on High Performance Computing Systems (HPCS'95)*, Montreal, Canada, July 1995, pp 304-315
5. Shatdal A. and J. F. Naughton, "Processing aggregates in parallel database systems", *Computer Sciences Technical Report # 1233*, Computer Sciences Department, University of Wisconsin-Madison, June 1994
6. Wolf J. L., Yu P. S., Turek J. and D. M. Dias, "A parallel hash join algorithm for managing data skew", *IEEE Transactions On Parallel and Distributed Systems*, Vol.4, No. 12, December 1993
7. Yan W. P. and P. Larson, "Performing group-by before join", in *Proceedings of the International Conference on Data Engineering*, 1994

A Deterministic Online Algorithm for the List-Update Problem

Hiranjoyti Mahanta and Phalguni Gupta

Department of Computer Science & Engineering
Indian Institute of Technology,
Kanpur 208 016, India pg@cse.iitk.ac.in

Abstract. An on-line algorithm for the *list update problem* has been proposed and is shown to be *2-competitive* under any sequence of access requests. The simpler version of the algorithm can be easily and efficiently implemented. Further, using this algorithm as a procedure in data compression technique it is possible to obtain better compression ratio than other known algorithms.

1 Introduction

List update problem is one of the popular problems which have been studied extensively in the area of on-line algorithms. The problem is to maintain a list of items (possibly by a linear linked list or a linear array) to serve the basic operations like *insert*, *access* and *delete*. Though there are some other data structures to maintain a dictionary of items to make *insert/delete* and *access* more efficient, in this problem by definition, we assume a representation typically similar to a linear linked list where access operations take time equal to the position of the item. For study of the problem in general, it is sufficient to consider the operation *access* only, as other operations viz. *insert* and *delete* can be reduced to *access* [6], [7]. *Paging* problem of memory management is a restricted version of this problem, also deterministic algorithms for list update problem can be used in the field of data compression [2].

After each operation, the items may be rearranged in order to make the future operations efficient; one way is to move the commonly requested items closer to the front of the list. Immediately after an *access* or *insert*, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. Any other exchanges that incur one unit cost per exchange are known as *paid exchanges*. The objective of the problem is to maintain the list in such a way that the total cost of *access* and *paid exchanges* is minimal.

The model of list update problem considered in this paper follows the *standard model* defined by Sleator and Tarjan in [7] also discussed in [6]. Here, $L = \{x_1, x_2, \dots, x_n\}$ is a set of items. We have an initial instance L_0 of the list containing items of L and $\sigma = \langle r_1, r_2, \dots, r_m \rangle$ is the requested sequence of access operations. A list update algorithm finds a service sequence $\omega = \langle E_1, E_2, \dots, E_m \rangle$

a series of rearrangements in which each E_i is a sequence of exchanges to be performed just before servicing r_i . An on-line algorithm does not have any knowledge of the future requests when servicing a request. Performance of an on-line algorithm is studied in terms of its competitiveness to an optimum cost off-line algorithm known as OPT. An on-line algorithm is *c-competitive* if

$$C_A(\sigma) \leq c.C_{OPT}(\sigma) + a \quad (1)$$

where $C_A(\sigma)$ denotes the cost incurred by the algorithm A to serve σ and $C_{OPT}(\sigma)$ denotes the cost paid by an optimal on-line algorithm OPT to serve the same request sequence σ ; and a is some constant. The algorithm is also called simply *competitive* if c in (1) is a constant [1].

In this paper we are going to propose a deterministic on-line algorithm for the list update problem. It is *2-competitive* and performs better than the existing best deterministic on-line algorithm *timestamp(0)* [2] on random data. A simpler version of the proposed algorithm is easier to implement and runs faster than many of the existing algorithms. We also show that our algorithm can be effectively used as a procedure in data compression techniques discussed in [4], [2] for achieving better compression ratio. Next section gives an overview of the existing related work in this area. Section 3 contains the new deterministic algorithm proposed by us. In Section 4 we have studied the behavior of our algorithm on a collection of data with respect to the existing algorithms.

2 Previous Work

Various on-line list update algorithms with their competitiveness and behavior on sample input are available today. There are few well known deterministic algorithms for the list update problem.

- *Move-to-Front*(MTF): The requested item is moved to the front of the list.
- *Transpose*(TRANS): The requested item is moved forward by one position by exchanging it with the immediately preceding item.
- *Frequency-count*(FC): It maintains frequency count for each of the items in the list. Whenever an item is accessed, increase the count by one.
- *Timestamp(0)*(TS(0)): It moves the accessed item say x via free exchanges to the position just before the first such item in the list which was accessed atmost once since the last access of x .

Sleator and Tarjan have formalized the problem by their standard model in [7]. Bentley et.al. in [3] have shown that MTF performs better than TRANS and FC in a collection of random input. Their results include the 2-competitiveness of MTF to the offline algorithm *Optimal static ordering* (is not overall optimal). Then they run these algorithms against some collection of documents and show that on the average MTF behaves better than others. Sleator and Tarjan in [7] have discussed some on-line algorithms in terms of competitiveness by amortization. It has been shown that on-line algorithm MTF is 2-competitive while TRANS and FC are not.

3 An Improved Algorithm

For a number of the analyses in the text we use the idea of considering decomposing the problem of n items to a number of lists of size 2, by taking all possible pairs of the list. For a list of n items $L = \{x_1, x_2, \dots, x_n\}$ and a request sequence σ , by σ_{ij} we denote the request sequence obtained from σ by considering the accesses only to the items of $\{x_i, x_j\}$.

Both MTF and TS(0) are competitive for any (worst case) input pattern. MTF is very much data sensitive; for a single occurrence of an item, it changes the list without considering the probability of occurrence of the item in near future. Moving an item of low frequency of occurrence to the front of the list creates overhead to the access of other items having higher frequency of occurrence. TS(0) changes the list slowly with respect to MTF - it does not move an item too forward whose frequency (local rather than global) hence probability of occurrence is low. But a keen observation to its behavior reveals that for the group of items with approximately equal recent frequency it does unnecessary exchanges for each alternate occurrences. As for example, on a list of two items $\{x, y\}$ and a request pattern $(xy)^l$ of length $2l$ TS(0) will do at least $2l - 3$ exchanges yielding a cost of at least $4l - 3$. In contrast, an OPT serves the same request pattern with $3l$ cost without doing any exchanges. This is because of the fact that a simple realization of OPT for a list of size 2 is - after each request, move the requested item via free exchange to the front if next request is also to the same item (see Proposition 4 of [6]). Here we propose an on-line algorithm *Slow Adapt* (SAD in short) which adapts with the changes in frequency of items slowly than TS(0) does, but it avoids unnecessary exchanges for the items of approximately equal recent frequencies. We show that SAD is 2-competitive to OPT as MTF and TS(0) are for any arbitrary input sequence. Then we show that overall behavior of SAD is better for random request sequence. Hence, SAD, in stead of TS(0), may increase the compression ratio when used in data compression algorithms.

The basic idea of SAD is to breakup the problem of n items in to $\frac{n(n-1)}{2}$ individual lists of 2 items each. Analyzing the list update problem pairwise for all the combinations already has been done in many articles [2], [3]. For each pair of items $\{x_i, x_j\}$ ($i < j$) SAD maintains a frequency function $f_{ij}(t)$. Though the function is defined as f_{ij} for $i < j$, we shall use f_{ij} and f_{ji} interchangeably to mean the same function. Value of each $f_{ij}(t)$ after serving $\sigma(t)$ indicates the recent access frequency of x_i with respect to that of x_j . We define $f_{ij(i < j)}(t)$ as follows:

Let, at time t_0 , any item of the pair $\{x_i, x_j\}$ is accessed for the first time, $f_{ij}(t_0)$ is set to -1 if the accessed item ($\sigma(t_0)$) appears before the other item of the pair, otherwise it is set to +1. For subsequent accesses,

$$\begin{aligned} f_{ij}(t) &= f_{ij}(t-1) - 2 \text{ if } \sigma(t) \in \{x_i, x_j\} \text{ and} \\ &\quad \sigma(t) \text{ appears before the other item in } \{x_i, x_j\} \\ &= f_{ij}(t-1) + 2 \text{ if } \sigma(t) \in \{x_i, x_j\} \text{ and} \\ &\quad \sigma(t) \text{ appears after the other item in } \{x_i, x_j\} \end{aligned}$$

$$= f_{ij}(t - 1) \text{ if } \sigma(t) \notin \{x_i, x_j\}$$

After each access to an item x_i , compute $f_{ij}(t) \forall j \in [1, n]$ and $j \neq i$. For a pair of items $\{x_i, x_j\}$ it is observable that the function $f_{ij}(t)$ will indicate the recent relative frequency. If the item appearing latter in the list is being accessed more than the function f_{ij} will have higher value. So, we put x_i before x_j through free exchanges if for any j , $f_{ij}(t)=3$. Exchange is optional if $f_{ij}(t) = 1$ and restricted if $f_{ij}(t) < 0$. Here, we are going to consider the variant in which we set $f_{ij}(t) = -1$ if $f_{ij}(t) = -3$ at anytime. This makes the algorithm easy to implement.

Lemma 1. *On a list of two items SAD brings the accessed item to the front of the list if and only if the last two requests were to the same item.*

The exchange rule: While maintaining a list with n items, after an access to x_i it is moved via free exchanges to the place just before the first such item x_k for which $f_{ik}(t)=3$. For all j such that x_j was exchanged with x_i , set $f_{ij}(t) = -f_{ij}(t)$.

Lemma 2. *The exchange rule described above obeys SAD for each pair of items.*

Now we proceed towards the proof of 2-competitiveness of the algorithm. We have seen that following the exchange rule described above, we can maintain SAD for each of the pairs in the list. Theorem 1 gives the inequality for each such pair of items. Later we sum up the inequality over all pairs to get relation given by the Theorem 3. Observe that this summing up is allowed because, SAD for n items maintains the relative ordering of SAD for each pair too. But, OPT can not always maintain the optimal order of each pair while serving for n items.

We consider any pair $\{x, y\} \subset L$ and the corresponding request sequence σ_{xy} . Then we can show that

Theorem 1. *For a list of two items $\{x, y\}$, and a request subsequence σ_{xy} , the following inequality holds: $2C_{OPT}(\sigma_{xy}) - C_{SAD}(\sigma_{xy}) \geq 1$*

Combining the result from pairwise analysis altogether, we obtain the relation for the complete list of n items. Here, one thing has to be observed that SAD maintains the relative ordering as in pairwise analysis while serving for the list of n items. So, we can sum up the cost due to SAD in pairwise lists to obtain the cost on the complete list. This follows the following claim.

Claim. The service sequence due to SAD on a list L of n items can be broken up into $\frac{n(n-1)}{2}$ service sequences of SAD on lists of 2 items, taking all possible pairs of items from the list of n items. Then

$$C_{SAD}(L, \sigma) = \sum_{\substack{x, y \in L \\ x \neq y}} C_{SAD}(\langle x, y \rangle, \sigma_{xy})$$

The same is not true for the optimal algorithm OPT. This is so because OPT on list of n items can not maintain the same order as in each of the lists of 2 items for all possible pairs. The following theorem makes it clear.

Theorem 2. *If $L_0 = \langle x_1, x_2, \dots, x_n \rangle$ is a list, $\sigma = \langle r_1, r_2, \dots, r_m \rangle$ is a request sequence and $C_A(L, \sigma)$ denotes the cost of servicing σ by algorithm A on an initial list L, then any off-line algorithm A' to serve σ on L_0 requires a cost*

$$C_{A'}(L_0, \sigma) \geq \sum_{i=1}^{n-1} \sum_{j=i+1}^n C_{OPT}(\langle x_i, x_j \rangle, \sigma_{x_i, x_j}) - m(n-2)$$

Using the above results we can conclude

Theorem 3. *SAD is 2-competitive.*

4 Experimental Results

Here we have followed the work of Albers et.al. by putting SAD for list rearrangement and compare its performance with MTF and TS(0). We have not implemented a complete data compression tool. However, we count the number of bits required to represent the accessed characters (or words in case of word-level compression) by integer representation scheme in [5]. Our experiment shows that SAD performs better than TS(0) and SAD runs faster than TS(0). In [4] it is shown that TS(0) yields a better compression ratio than that of MTF. Thus, we conclude that adopting SAD than TS(0) is preferable for at least this class of problems.

References

1. Albers,S.: Competitive Online Algorithms. BRICS Lecture Series (1996)
2. Albers, S., Mitzenmacher, M.: Average case analysis of list update algorithms, with applications to data compression. Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 1099, Springer-Verlag, Berlin Heidelberg New York (1996) 514-525 682-693
3. Bentley, J. L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Communications of the ACM 28(1985) 404-411
4. Bentley,J. L., Sleator, D. S., Tarjan, R. E., Wei, V. K.: A locally adaptive data compression scheme. Communications of the ACM 29 (1986) 320-330
5. Elias, P.: Universal codeword sets and the representation of the integers. IEEE Trans on Information Theory 21 (1975) 194-203
6. Reingold, N., Westbrook, J.: Off-line algorithm for the list update problem. Information Processing Letters 60 (1996) 75-80
7. Sleator, D. D., Tarjan, R. E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28 (1985) 202-208

Session III-B

Mobile Computing - I

Chair: Sajal Das,

University Of North Texas

Link-State Aware Traffic Scheduling for Providing Predictive QoS in Wireless Mobile Multimedia Networks

AZM Ekram Hossain and Vijay K. Bhargava, FIEEE

Department of Electrical and Computer Engineering
University of Victoria, PO Box 3055 STN CSC
Victoria, BC V8W 3P6, Canada
Tel: +1-250-721-8617, Fax: +1-250-721-6048
{ekram, bhargava}@ece.uvic.ca

Abstract. Performance of a centralized traffic priority based dynamic burst level cell scheduling scheme is investigated in a correlated fading channel. The scheduling scheme is designed for transmission of multi-service traffic over TDMA/TDD channels in a WATM (Wireless ATM) network. In this scheme, the number of slots allocated to a VC (Virtual Circuit) is changed dynamically depending on the traffic type, system traffic load, the TOE (Time of Expiry) value of the data burst and data burst length in addition to the channel error status. In fact, the scheme can be considered as a single integrated scheme for link-level traffic scheduling and ARQ-based error control for multiservice wireless networks. Performance of the proposed scheme is evaluated through computer simulation for realistic voice, video and data traffic models, the QoS requirements of different traffic classes in a wireless mobile network and correlated Rayleigh fading channel model. Simulation results show that, the proposed scheduling scheme can provide reasonably high channel utilization with predictive QoS guarantee in a multiservice traffic environment. At the same time such a scheme can result in an energy efficient TDMA/TDD medium access control protocol for broadband wireless access.

1 Introduction

With increased focus on multimedia networking services and ubiquitous information access, ‘multimedia migration’ to mobile portable platforms seems to be the most significant issue in the next generation wireless mobile networks. It is expected that with mobility “built-in” to future network API (Application Programming Interface) specifications, uniform fixed/wireless network API will make terminal mobility transparent to application software[1]. ATM-based technology is being envisioned to provide high speed wireless multimedia communications. In fact, the fine grain multiplexing provided by ATM due to the fixed small cell size is well suited to slow-speed wireless links since it leads to lower delay jitter and queueing delays. In addition, the QoS specifiable VCs

in the ATM provide the ability to meaningfully distinguish the data cells sent over the air and not treat all of them according to some generic policy[2]. The different traffic streams with the different QoS requirements in a multimedia flow can be transported over different VCs. The network level QoS requirements for multimedia data which are derived from their transport characterization are specified by parameters such as *CTD* (Cell Transfer Delay), *CDV* (Cell Delay Variation) and *CLP* (Cell Loss Probability). For nrt-VBR, ABR and UBR traffic, *CLP* is the most important QoS parameter to be guaranteed. For CBR and rt-VBR traffic, both *CLP* and *CTD* are to be guaranteed, and if it is assumed that the *CDV* can be compensated by using buffering and output control at the BS(Base Station)/AP(Access Point), it gets merged with the *CTD* issue.

Providing predictive QoS in a wireless mobile multimedia network has the advantage of not requiring a call to specify its traffic parameters, but the call must belong to one of a predefined set of classes and its traffic should correspond to the traffic characteristics of that class if the guarantees are to be reliable. Measurement-based bandwidth allocation and call admission control[3] combined with the relaxed predictive QoS commitment can achieve high level of network utilization. In a wireless mobile network, predictive QoS can be ideal for scalable multimedia applications which can tolerate some occasional degradations in network performance due to channel errors and user mobility. This can be achieved by measurement-based call admission control and certain degree of bandwidth reservation combined with traffic priority based dynamic bandwidth allocation in the MAC (Media Access Control) protocol.

Earlier we investigated a set of centralized dynamic priority based burst level cell scheduling schemes, namely, FCFS-FR (First Come First Served with Frame Reservation), FCFS-FR+, EDF-FR (Earliest Deadline First with Frame Reservation), EDF-FR+ and MTDR (Multitraffic Dynamic Reservation) for WATM for transmitting multiservice traffic over TDMA/TDD channels where the number of slots allocated to a VC is changed dynamically depending on the traffic type, system load and the TOA (Time of Arrival)/TOE (Time of Expiry) values of the data bursts[4]. Simulation results showed that, EDF-FR+ and MTDR schemes provide tolerable QoS guarantee with reasonably high channel utilization in a multiservice traffic environment and the EDF-FR+ scheme was found to provide better multiplexing performance than the MTDR scheme. In this paper we explore the performance of EDF-FR+ scheme under correlated channel errors. Information about channel error correlations is exploited so that it becomes robust to channel impairments to some extent. In a mobile radio environment, high speed data transmission (e.g, 20 *Mbps* in 5 *GHz* band), will suffer significantly from correlated errors in a slowly varying fading channel. Scheduler performance can be improved based on channel prediction and by not granting transmit permission to users whose channel is bad. We refer to the the enhanced EDF-FR+ scheduling policy employing some channel prediction as CSAS (Channel State Aware Scheduling) scheme. We evaluate the performance of the CSAS scheme by computer simulation in a Rayleigh fading channel. The effects of bursty channel errors on the QoS degradation of multiservice traffic are

determined in terms of long term cell loss rate and average cell transfer delay under the CSAS scheme.

The EC-MAC (Energy Conserving MAC) protocol proposed in [5] uses a 'priority round-robin with dynamic reservation update and error compensation' scheduling where CBR and VBR traffic are prioritized over UBR traffic and within the same traffic type different connections are treated using round-robin mechanism. TOE values of data bursts are not explicitly taken into account for scheduling. Our work is different in that the scheduling scheme considered in this paper is based on resource isolation for different types of traffic and at the same time dynamic allocation based on the instantaneous needs of the connections carrying the same type of traffic. The priority of a connection is based on the traffic type, the TOE value of the corresponding data burst and the data burst length. The idea is to reserve some bandwidth (which is changed dynamically) during each frame time for each class of traffic and to distribute the excess bandwidth according to the instantaneous needs of the admitted connections. A round-robin based link-level channel-state-dependent packet scheduling (CS-DPS) was proposed in [6] mainly as a means of enhancing TCP performance for a system featuring RTS (Request to Send)-CTS (Clear to Send) messaging. This scheme was enhanced by combining it with CBQ (Class-Based Queueing) in [7] though the different QoS requirements of multimedia traffic were not considered. In fact, in a WATM network per cell scheduling may not be computationally feasible and per-VC scheduling is more desirable from implementation point of view. Moreover, the RTS-CTS based channel probing used in the above works may often result in discrepancy between the actual channel state and the estimated channel state since links are monitored only occasionally (frequency of which is determined by a parameter g). In a WATM network with TDMA/TDD-based MAC protocol considered in this paper, each MT suffering collision can send a 'probe message' in a minislot during the RA period in each frame time which is typically very small. If the BS/AP receives it correctly, it can resume allocating bandwidth. Thus it is possible to sample the channel status in a more 'fine-grained' manner in this case by using an efficient contention resolution scheme in the RA minislots and consequently a better channel estimation may be achieved.

2 The CSAS Algorithm

The TDMA/TDD frame structure, the media access procedure, the protocol parameters and their notations are same as those described in [10]. If a frame is lost due to channel errors, the corresponding cells are kept waiting in the transmit data buffer until they are successfully transmitted or their waiting time exceeds the corresponding TOE value. Let $P_{CBR}(t)$, $P_{rt-VBR}(t)$ and $P_{nrt-VBR}(t)$ denote the set of 'active' CBR VC, rt-VBR VC and nrt-VBR VC respectively during frame time t for which the channel conditions are predicted to be good. Let n_{CBR} , n_{rt-VBR} and $n_{nrt-VBR}$ denote the corresponding cardinalities and let $P(t) = P_{CBR}(t) \cup P_{rt-VBR}(t) \cup P_{nrt-VBR}(t)$. The VC identifiers are arranged in

the ascending order of their TOE values and among the VCs with the same TOE value the one with the longest queue-length precedes the others. When the uplink traffic corresponding to a VC is not ACKed by the BS/AP, the corresponding MT assumes that the cells have been lost and it starts transmitting 'probe' messages periodically during *RA* period. A few of the *RA* minislots can be reserved for these 'probe' messages. As soon as the BS/AP receives one of these probe messages correctly, it resumes scheduling the traffic corresponding to these VCs again for transmission.

$VC_i(CBR)$ s are given priority over $VC_i(rt-VBR)$ s due to their more stringent delay requirements. A minimum amount of bandwidth is reserved in each frame for audio, real-time video and non real-time traffic by setting a nonzero value for R_{CBR} , R_{rt-VBR} and $R_{nrt-VBR}$ respectively. The values of the parameters R_{CBR} , R_{rt-VBR} and $R_{nrt-VBR}$ can be varied adaptively depending on the traffic load and QoS satisfaction of the 'active' VCs using some measurement-based scheme. The maximum and minimum values of these parameters will be dependent on the admission control and pricing policies for CBR, rt-VBR and nrt-VBR VCs. The simplest adaptation policy is to toggle them between some fixed value of $R_{CBR}/R_{rt-VBR}/R_{nrt-VBR}$ and 0 depending on whether there is any 'active' CBR/rt-VBR /nrt-VBR VC when scheduling computations are performed.

The scheduling algorithm is as follows:

```

1. if ( $n_{CBR}(t) = 0$ ) {
     $r_{CBR}(t) = 0$ ,  $Z(t) = R_{CBR}$ 
}
else {
     $r_{CBR}(t) = R_{CBR}$ ,  $Z(t) = 0$ ,  $i = 0$ 
    while (  $i < |P_{CBR}(t)|$  and  $r_{CBR}(t) > 0$  ) {
        allocate  $\min(Q_i(t), r_{CBR}(t))$  slots to  $VC_i$  in  $P_{CBR}(t)$ 
         $r_{CBR}(t) = r_{CBR}(t) - \min(Q_i(t), r_{CBR}(t))$ 
         $i++$ 
    } //while
     $Z(t) = Z(t) + r_{CBR}(t)$ 
} //else
2. if ( $n_{rt-VBR}(t) = 0$ ) {
     $r_{rt-VBR}(t) = 0$ ,  $Z(t) = Z(t) + R_{rt-VBR}$ 
}
else {
     $r_{rt-VBR}(t) = R_{rt-VBR}$ ,  $i = 0$ 
    while (  $i < |P_{rt-VBR}(t)|$  and  $r_{rt-VBR}(t) > 0$  ) {
        allocate  $\min(Q_i(t), r_{rt-VBR}(t))$  slots to  $VC_i$  in  $P_{rt-VBR}(t)$ 
         $r_{rt-VBR}(t) = r_{rt-VBR}(t) - \min(Q_i(t), r_{rt-VBR}(t))$ 
         $i++$ 
    } //while
     $Z(t) = Z(t) + r_{rt-VBR}(t)$ 
} //else

```

```

3. if ( $n_{nrt-VBR}(t) = 0$ ) {
     $r_{nrt-VBR}(t) = 0$ ,  $Z(t) = Z(t) + R_{nrt-VBR}$ 
}
else {
     $r_{nrt-VBR}(t) = R_{nrt-VBR}$ ,  $i = 0$ 
    while (  $i < |P_{nrt-VBR}(t)|$  and  $r_{nrt-VBR}(t) > 0$  ) {
        allocate  $\min(Q_i(t), r_{nrt-VBR}(t))$  slots to  $VC_i$  in  $P_{nrt-VBR}(t)$ 
         $r_{nrt-VBR}(t) = r_{nrt-VBR}(t) - \min(Q_i(t), r_{nrt-VBR}(t))$ 
         $i++$ 
    } //while
     $Z(t) = Z(t) + r_{nrt-VBR}(t)$ 
} //else
4.  $i = 0$ 
while (  $i < |P(t)|$  and  $Z(t) > 0$  ) {
    allocate  $\min(Q_i(t), Z(t))$  slots to  $VC_i$  in  $P(t)$ 
     $Z(t) = Z(t) - \min(Q_i(t), Z(t))$ 
     $i++$ 
} //while

```

3 Simulation Environment, Channel Model, Traffic Models and Performance Measures

A C-coded event driven simulator is used. The simulator is run for 2 *million* frames (with the parameters shown in Table 1) which correspond to 1320 *seconds* for a frame size of 30 *slots*. Simulation statistics was flushed after 500000 *frames* to avoid the transient effects. Since the performance of the scheduler is of major concern, we disregard the contentions among the reservation requests during the *RA* subperiods. SR-ARQ (Selective-Repeat ARQ)-based error control scheme is assumed.

For a broad range of parameters, the frame (i.e., a data block of duration T) error process in a mobile radio channel where the multipath fading can be considered to follow a Rayleigh distribution can be modeled using a two-state Markov chain with transition probability matrix M_c [9]: $\begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix}$ where p and $1-q$ are the probabilities that the j -th frame transmission is successful, given that the $(j-1)$ -th frame transmission was successful or unsuccessful, respectively. We assume that during the time period T , the fading envelope does not change significantly. For each of the MTs, the channel evolution in each TDMA/TDD frame is assumed to be independent and determined by the two parameters p and q which in turn are determined by f_dT ¹, the normalized Doppler bandwidth and P_E , the average frame error probability. P_E describes the channel quality in terms of *fading margin* F ². Different values of P_E and f_dT result in different

¹ where $f_d = v/\lambda$ = mobile speed/carrier wavelength

² the maximum fading attenuation which still allows correct reception

degree of correlation in the fading process. When f_dT is small, the fading process is very correlated; on the other hand, for large values of f_dT , the fading process is almost independent. The variation of expected length of error burst with P_E is shown in Fig. 1 for several values of terminal mobility assuming a carrier frequency of 5 GHz, channel data rate of 20 Mbps and a frame length of 30 cells each of size 55 bytes. For a certain value of average frame error rate P_E (particularly for a value in the range of 10^{-2} to 10^{-3}), the error burst length increases as user mobility (and hence f_dT) decreases.

All the cells comprising a burst have the same TOE value. For the nrt-VBR traffic type, data bursts are generated only either after the current burst has been transmitted or it has ‘expired’. Traffic is assumed to be symmetric in both the forward and the reverse link. The amount of available buffers at an MT is assumed to be infinite. A vocoder with fast speech activity detector, a GBAR (Gamman-Beta Autoregressive) model and an LRD (Long Range Dependent) model are used as CBR, rt-VBR and nrt-VBR traffic sources respectively[4].

The QoS parameters considered are CLP_{CBR} (Average CBR Cell Loss Probability), CTD_{CBR} (Average CBR Cell Transfer Delay), CLP_{rt-VBR} (Average rt-VBR Cell Loss Probability), CTD_{rt-VBR} (Average rt-VBR Cell Transfer Delay), $CLP_{nrt-VBR}$ (Average nrt-VBR Cell Loss Probability), $T_{nrt-VBR}$ (Average nrt-VBR Throughput) and U (Channel Utilization)[10]. It is assumed that predictive QoS can be provided to the voice, video and data traffic if the values of CLP_{CBR} , CLP_{rt-VBR} and $CLP_{nrt-VBR}$ are of $O(10^{-2})$, $O(10^{-3})$ and $O(10^{-6})$ respectively and the maximum values CTD_{CBR} and CTD_{rt-VBR} are less than 10ms and 15ms respectively. U is calculated without taking control overheads into account.

Table 1. Simulation parameters

Parameter	Value	Parameter	Value
Channel rate	20 Mbps	Slot size	$22\mu s$
No. of ACK slots	1	Principal talkspurt length	1.00 s
Principal silence gap length	1.35 s	Mini-spurt length	0.275 s
Mini-gap length	0.05 s	Voice coding rate	32 Kbps
Mean no. of cells/video frame	104.9	Variance of no. of cells/video frame	882.09
Correlation coefficient, ρ	0.984	Video interframe rate	40ms
Mean nrt-VBR burst length	0.48 KB	nrt-VBR source Pareto shape parameter	1.1
Mean nrt-VBR burst interarrival time	1s	TOE of a voice cell	32 ms
TOE of a video cell	40 ms	TOE of a data cell	1 s

4 Simulation Results

rt-VBR traffic only: The variation of U and CLP_{rt-VBR} with N_{rt-VBR} in a rt-VBR traffic only scenario is shown in Fig. 1. For $P_E = 0.01$ (i.e., channel fading margin = 19.978 dB) and $f_dT = 0.00308$ which corresponds to user velocity of 1 km/hr, for carrier frequency equal to 5 GHz, $CLP_{rt-VBR} \approx 1 \times 10^{-3}$

and $U \approx 0.84$ when $N_{rt-VBR} = 6$. For the same value of P_E and $f_d T$, with $N_{rt-VBR} = 7$, $CLP_{rt-VBR} \approx 7 \times 10^{-3}$ and the resulting channel utilization is about 0.95. As long as U is below certain limit (e.g., 0.84), the scheme can provide tolerable loss and delay performance for P_E as high as 1%. Since with low user mobility the channel fading events are more correlated, for a certain value of P_E , as v decreases, CLP_{rt-VBR} increases.

CBR and rt-VBR traffic: In case of CBR and rt-VBR traffic mix with $R_{rt-VBR} = 4$ and $R_{CBR} = 0$, for $P_E = 0.01$ and $v = 1 \text{ km/hr}$, $CLP_{CBR} \approx 1 \times 10^{-2}$ and $CLP_{rt-VBR} \approx 7.5 \times 10^{-4}$ when $N_{rt-VBR} = 5$ and $N_{CBR} = 30$ (Fig. 2). In this case $U \approx 0.75$. As N_{rt-VBR} is increased to 50, $CLP_{rt-VBR} \approx 1.8 \times 10^{-3}$. In this case $U \approx 0.77$ and CTD_{CBR}/CTD_{rt-VBR} are well below the assumed QoS limits. It is observed that increasing the value of R_{rt-VBR} causes large degradation in CLP_{CBR} . The average cell loss and delay measures and channel utilization depend on the proportion of each traffic in the total traffic load and the value of the reservation parameters. It is observed that as the proportion of CBR traffic decreases, channel utilization increases with a consequent increase in the throughput of rt-VBR traffic.

CBR, rt-VBR and nrt-VBR traffic: In case of CBR, rt-VBR and nrt-VBR traffic mix, channel utilization of about 0.75 can be achieved with tolerable QoS for all traffic streams when $N_{CBR} = 25$ and $N_{rt-VBR} = 5$ for P_E as high as 0.01 (Fig. 3). In fact, due to very large variance in the nrt-VBR burst size (resulting from the Pareto distribution) it may not be possible to meet the very low loss requirement of nrt-VBR traffic and at the same time achieve high channel utilization when the delay of the nrt-VBR traffic is to be bounded. By limiting the burst size to a maximum value, better channel utilization can be achieved. Channel utilization also depends on the ratio of the real-time traffic-load to total traffic load. It has been observed that as the ratio decreases, channel utilization increases and consequently $T_{nrt-VBR}$ increases. In case of non-LRD traffic channel utilization increases significantly. Some simulation results are shown for short-range dependent nrt-VBR traffic. In this case it is assumed that burst lengths are exponentially distributed with mean duration of 0.1s with data rate of 1 Mbps and maximum tolerable delay of a burst is 3 s. With $N_{CBR} = 40$ and $N_{rt-VBR} = 5$, $U \approx 0.90$ and $CLP_{nrt-VBR} < 10^{-6}$ in case of non-LRD nrt-VBR traffic. In this case $T_{nrt-VBR} \approx 2.7 \text{ Mbps}$.

5 Outlook

For multiplexing bursty multimedia traffic (which are often difficult to model) over WATM air interface, traffic priority based dynamic cell scheduling schemes are viable to achieve high channel utilization and at the same time to provide predictive QoS to different traffic streams. The burst level cell scheduling scheme described in this paper for TDMA/TDD-based wireless access is suitable for providing tolerable QoS in a multitraffic environment. In addition, establishing synchronization at the BS/AP is expected to be relatively simpler since during each frame time it is more likely that fewer number of MTs are involved in

transmission/reception. The reservation parameters, namely R_{CBR} , R_{rt-VBR} and $R_{nrt-VBR}$ are the ‘tuning knob’s that can be dynamically adjusted for varying the QoS required for the different traffic streams. Simulation results show that, for a TDMA/TDD frame size of $0.66ms$ and link speed of $20\ Mbps$, reasonably good QoS (in terms of long term cell loss rate and cell transfer delay) can be obtained for bursty voice, GBAR video and LRD data traffic even with a simple binary adaptation policy. Improved performance may be achieved by employing some measurement-based scheme to adjust the values of R_{CBR} , R_{rt-VBR} and $R_{nrt-VBR}$ according to the instantaneous traffic load and the QoS satisfaction of the different VCs but at the expense of more processing overhead. We also observe that providing bounded delay to LRD traffic and at the same time achieving very high channel utilization may not be possible. Higher channel utilization is achieved for short-range dependent data traffic.

The scheme can be easily adapted as a MAC policy for wireless access to the emerging DS Internet by integrating with some traffic conditioning schemes at the MTs. In fact the concept here is in line with the recently proposed Assured Forwarding (AF) Per-Hop Behavior (PHB) for DS Internet[11]. The BS/AP functioning as an ingress node to the wireline network can integrate the traffic admission policy with the wireless MAC protocol. The BS/AP scheduler can also use ECN (Explicit Congestion Notification)[12] information from downstream routers for scheduling data bursts at the MTs. Traffic conditioning at the network edge (i.e., in MTs) along with a centralized scheduling scheme can provide predictable QoS to multimedia traffic in a DS wireless network.

References

1. Raychaudhuri, D.: Wireless ATM: An Enabling Technology for Multimedia Personal Communication. *Wireless Networks*, vol. 2, 1996, 163–171
2. Agrawal, P. et al.: SWAN: A Mobile Multimedia Wireless Network. *IEEE Personal Communications Magazine*, Apr. 1996, 8–33
3. Jamin, S., Danzig, P. B., Shenker, S. J., Zhang, L.: A Measurement-Based Admission Control Algorithm for Integrated Services Packet Networks. *IEEE/ACM Transactions on Networking*, February 1997, URL: <http://netweb.usc.edu/jamin/admctl/ton96.ps.Z>
4. Hossain, AZM. E., Bhargava, V. K.: Link-Level Traffic Scheduling for Providing Predictive QoS in Wireless Multimedia Networks. submitted to *IEEE Transactions on Multimedia*
5. Chen, J.-C., Sivalingam, K.M., Agrawal, P., Acharya, R.: Scheduling Multimedia Services in a Low-Power MAC for Wireless and Mobile ATM Networks. *IEEE Transactions on Multimedia*, vol. 1, no. 2, June 1999, 187–201
6. Bhagwat, P., Bhattacharya, P., Krishna, A., S. Tripathi, S.: Enhancing Throughput over Wireless LANs Using Channel State Dependent Packet Scheduling. Proc. *IEEE INFOCOM*, 1996, vol. 3, 1133–1140, URL: <http://www.cs.umd.edu/~pravin/publications/publist.htm>
7. Fragouli, C., Sivaraman, V., Srivastava, M. B.: Controlled Multimedia Wireless Link Sharing via Enhanced Class-Based Queuing with Channel-State-Dependent Packet Scheduling. Proc. *IEEE INFOCOM*, 1998, 572–580, URL: <http://www.cs.ucla.edu/~vijay/>

8. Acampora, A.: Wireless ATM: A Perspective on Issues and Prospects. IEEE Personal Communications Magazine, Aug. 1996, 8–17
9. Chockalingam, A., Zorzi, M., Milstein, L. B. and Venkataram, P.: Performance of a Wireless Access Protocol on Correlated Rayleigh Fading Channels with Capture. IEEE Transactions on Communications, vol. 46, no. 5, May 1998, 644–655
10. Hossain, AZM. E., Bhargava, V. K.: Scheduling Multiservice Traffic for Wireless ATM Transmission over TDMA/TDD Channels. Proc. IEEE GLOBECOM, 1999
11. Heinanen, J., Baker, F., Weiss, W. and Wroclawski, J.: Assured Forwarding PHB Group. Internet Draft - work in progress, <http://www.ietf.org/internet-drafts/draft-ietf-diffserv-af-06.txt>
12. Kalyanaraman, S., Harrison, D., Arora, S., Wanglee, K. and Guarriello, G.: A One-bit Feedback Enhanced Differentiated Services Architecture. Internet Draft - work in progress, <http://www.ietf.org/internet-drafts/draft-shivkuma-ecn-diffserv-01.txt>

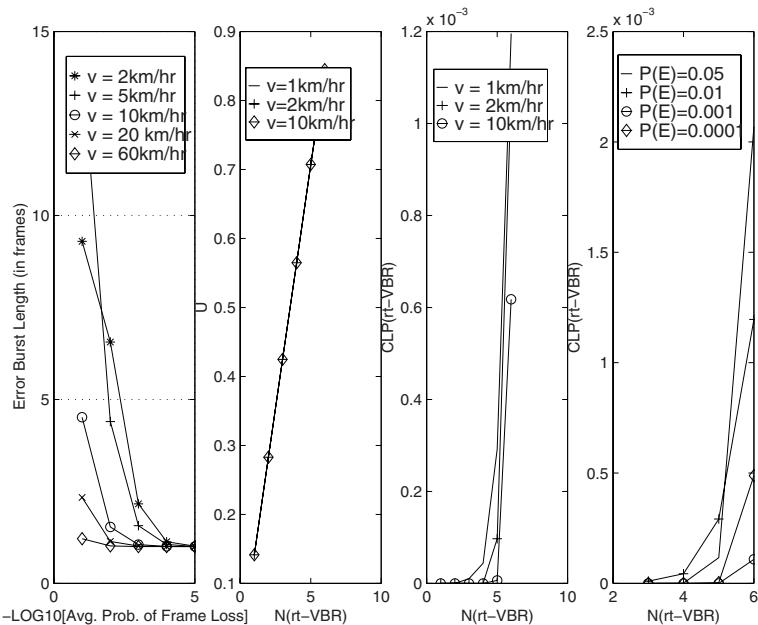


Fig. 1. Variation of average burst error length with P_E and Performance in single-traffic (rt-VBR only) scenario

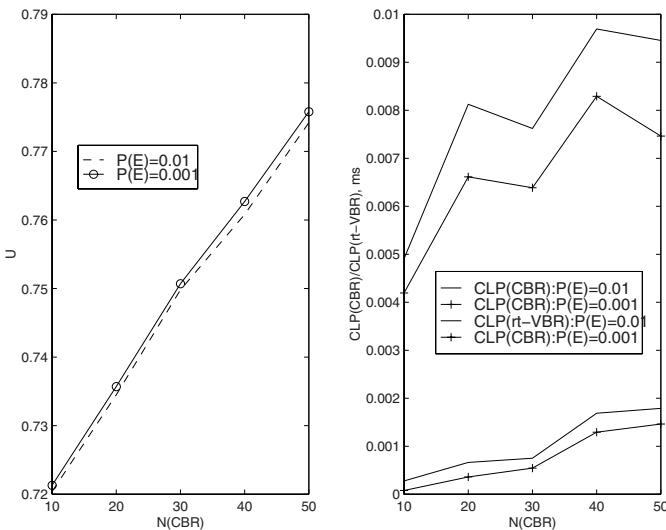


Fig. 2. Performance in a multitraffic (CBR and rt-VBR) scenario (for $N_{rt-VBR} = 5, R_{rt-VBR} = 4, R_{CBR} = 0$)

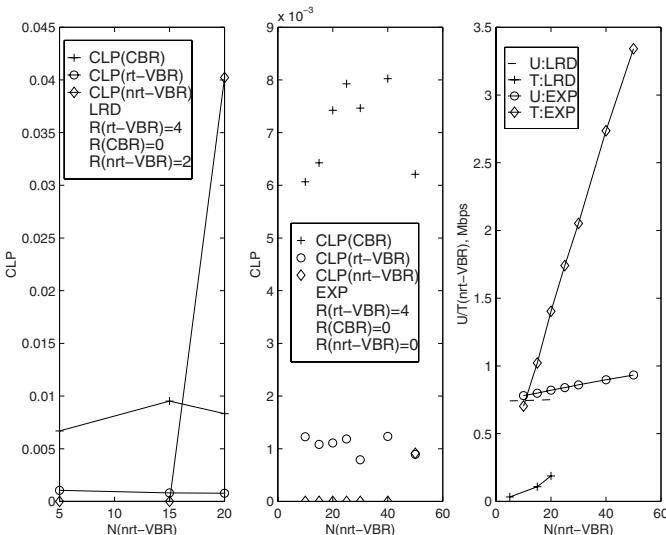


Fig. 3. Performance in a multitraffic (CBR, rt-VBR and nrt-VBR) scenario for LRD and non-LRD nrt-VBR (shown as EXP) traffic

Enhancing Mobile IP Routing Using Active Routers

Kwan Wu Chin¹, Mohan Kumar¹, and Craig Farrell²

¹ School of Computing, Curtin University of Technology,
Bentley, Western Australia, 6152
chinkw.kumar@cs.curtin.edu.au

² NDG Software, Suite 200, 12680 High Bluff Drive,
San Diego CA 92130 USA
craig@ndgsoftware.com

Abstract. This paper investigates how active networking technology can be applied to enhance Internet Engineering Task Force (IETF) Mobile IP. By leveraging the intra-network computation, packet redirection ,handoff and sending of binding updates is performed by active routers (ARs). Given these enhancements we compared the performance of transmission control protocol (TCP) on mobile IP, optimized mobile IP and our ARs scheme. Our results show that ARs allow TCP to recover faster and achieve better throughput due to low-latency handoff.

1 Introduction

The proliferation of wireless computing within modern networking environments have seen many problems emerge, in particular mobile host (MH) routing. Mobile IP [1] specifically addresses routing between MHs on top of the existing IP protocol.

The main problems in Mobile IP [1] are:

1. Triangle Routing. This problem arises when the MH is in a foreign network. All packets destined for the MH need to be redirected via its HA. This increases the delay in delivering packets and wastes bandwidth.
Optimised Mobile IP [2] solves the triangle routing problem by sending a binding update to a CH thus reducing latency. In addition, a binding update is sent to the previous foreign agent (FA). This enables the previous FA to tunnel packets to the MH's current care-of-address.
2. Packet Loss. The Mobile IP protocol specification dictates that the previous FA is not notified of the MH's new care-of-address. As a result, the old FA discards any packet which is not addressed to MHs in its visitor list [1]. As a consequence of the packet discards, transport protocols such as TCP invoke congestion control procedures which reduce throughput [3].
3. Slow binding updates. In the optimised Mobile IP [2] the mobility binding update time for the CH is a function of the time taken for the registration request to reach the HA and the RTT from the HA to the CH. The processing

delay encountered at the HA is also a minor factor. Within this time period, the CH continues to send packets to the MH's old location. The number of packets being forwarded is a function of the data rate between the FH and the MH's previous location. In a gigabit network environment this means a significant portion of the application's packets are being misdirected. This results in inefficient use of bandwidth. Apart from that misdirected packets need to be buffered at the previous FA. This is not scalable, considering an environment with high number of MHs.

4. Slow handoff time. The registration reply is a function of the RTT from the HA to the MH. If the MH's HA is at the other end of the globe, handoff takes considerably longer.

To address the above mentioned problems we leverage the processing capabilities of routers in active networks (ANs) [4]. The AN paradigm allows end-nodes to program network elements so that user specific computation can be performed. The programs can extend the capabilities of the network element(s) or provide customised computation on a set of packets (i.e per flow computation) [4]. There are two proposed methods for computation at network elements, programmable switches [5] and the capsule approach [6].

2 Active Routers

ARs are routers which allow customisation of user traffic. They contain predefined routines which may be loaded on a per-connection basis. These routines may run until they are specifically unloaded or until their time of service expires¹. The main operation of ARs is to monitor packets (including encapsulated packets) going to and from the MH and invoke specific functions to customise the flow of the given traffic. In our scheme, ARs are programmed² by the HA during connection setup and each AR maintain a security association with the HA. ARs acquire the current care-of-address for MH's through the registration request messages sent to the HA by the MH. By monitoring packets addressed to the MH, packets which are addressed to the MH's old care-of-address can be redirected. Apart from that ARs also maintain a list of CH addresses that are communicating with the MH. During handoff binding updates are sent to CH by ARs. As a result CHs receive binding updates faster. Thus traffic can be sent towards the MH sooner compared to the optimized Mobile IP scheme [2] where binding updates are issued by the HA.

In our scheme the following actions are taken by ARs upon receipt of messages generated by Mobile IP for a given connection:

- **Registration Requests from the MH.**

This message contains the MH's current care-of-address and is sent to the HA after handoff. The registration request is also used by the MH to renew

¹ e.g. When the current connection has been idle for a given time

² Only one program is loaded per MH

its registration request after the set lifetime has expired and to deregister itself when it returns home [1]. When an AR intercepts this request, it authenticates the request and records the new care-of-address and the requested lifetime of the request before forwarding the registration request. The lifetime is used by the ARs to expire old information. All ARs along the route record the MH's care-of-address. If an AR has a record of CHs communicating with the MH then binding updates are sent. The AR containing reachability to the MH's old and new BSs is termed the AR-CX. AR-CX is responsible for sending registration reply to the MH and programs ARs from it to the MH's new location. In the worst case scenario, the AR-CX is the AR local to the HA. In this case, the AR scheme will converge to the route optimisation Mobile IP scheme [2]. Once the ARs are programmed any packets from CHs are redirected and binding updates sent.

- **Registration Reply.**

In our ARs scheme, registration reply is generated by both the HA and AR-CX. The reply from the HA is forwarded to the MH. All ARs along the route authenticate the reply using the 64-bit number identification [1] in the registration reply message and record the status of the reply.

In the case where no registration reply is received after the registration request has been sent, ARs are responsible for retransmitting the registration request to the HA until a registration reply is received ³.

- **Binding Warning.**

This is generated by a node upon receiving a packet tunneled to an out-of-date care-of-address [2]. If an AR has an updated care-of-address the AR sends a binding update to the node which tunneled the packet.

- **Binding Update.**

Upon receiving a registration request from the MH, the AR-CX generates binding updates to the previous FA and any CHs. Note that only the AR-CX generate a binding update to the previous FA.

3 Simulation Methodology

We validate our scheme using simulation. TCP's performance is analysed and compared in three schemes: mobile IP, optimized mobile IP and active routers. The wired portion of the network has a speed of 10Mb/s and the wireless channel has a data rate of 2Mb/s and no bit errors. Bit errors are assumed to be handled by a data link layer protocol. The cell areas in our topology overlaps, which means there are no *silent* areas. The MH controls the time of handoff, hence the time of handoff and the duration of the handoff procedure can be measured. Each active router is capable of processing 5000 packets per second. For each simulation a persistent source transmits n packets of size 536 bytes from the CH to the MH.

³ Retransmission of registration request is specified in Section 3.6.3 in the Mobile IP specification [1]

4 Results

In this simulation the MH is set to migrate to a new cell at $t = 50.0$. The persistent source is set to transfer 80 packets each of size 536 bytes from the CH to the MH⁴.

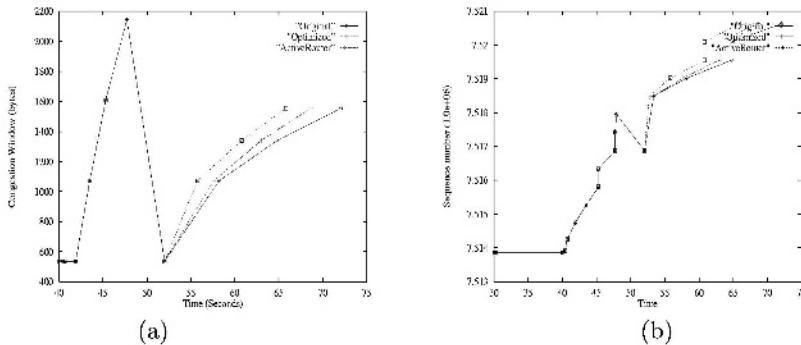


Fig. 1. These graphs show the congestion window size and sent sequence number. As can be seen the AR scheme causes TCP to recover faster after handoff at $t = 50.0$. As a result a higher throughput is observed.

From Figures 1(a) and (b) we see that ARs perform well above the mobile IP models since the CH and the MH no longer rely on the HA for the care-of-address binding update.

Figure 1 shows that TCP slow-start recovers fastest when the ARs are programmed with our scheme. A measure of the improvement is shown by the sequence numbers increasing faster in the ARs curve than it does with either the optimised or original Mobile IP curves.

Figure 1(a) also shows that Mobile IP (labelled Original) does not perform well since the CH needs to send each packet to the HA where it is tunneled to the MH. This causes unnecessary traffic which is inefficient particularly when the MH is located near the CH, meaning that longer RTT induced delays may occur. Moreover congested link to the HA can be avoided. The congestion window for Mobile IP increases at $t = 58.2$, a delay caused by TCP's exponential backoff. In the optimised Mobile IP, the congestion window starts increasing after $t = 57.6$ and for the ARs approach the congestion window advances at $t = 55.8$. This faster recovery enables the sender to increase its sequence number sooner which results in an overall improvement over the other Mobile IP models. The time taken to transmit 80 packets each of size 536K packets for the three models is different. The ARs scheme has the highest throughput due to faster recovery after handoff. In this simulation the CH gets updated with the new care-of-address

⁴ In our simulations no variable are dynamic or random, hence these simulations runs are repeatable. Furthermore we noticed similar behaviour given different handoff time.

	Mean	Standard Deviation
Original	1.43	1.0
Optimised	1.35	0.87
Active Router	1.29	0.77

Table 1. RTT mean and standard deviation. This shows the CH observed a lower mean RTT and lower deviation when the ARs scheme is used.

2.9ms after handoff for the ARs approach, and 9.1ms for the optimised Mobile IP. No binding update is sent to the CH in the original Mobile IP specification. Handoff takes 2.8ms using our approach and 7.4ms for the Mobile IP models.

5 Conclusion

The implications and potential performance improvements of ARs programmed with an active handoff algorithm have been analysed. Four main limitations of the Mobile IP models have been identified and our results showed that ARs can be used to alleviate these. ARs can provide an efficient and optimised routing scheme. Of importance is the CH being notified earlier of the MH's new care-of-address which enables it to tunnel packets along an efficient route sooner. Moreover significant deviation in RTTs which are inherent in the Mobile IP models was also minimised. This lower deviation means that TCP was less likely to invoke slow-start congestion avoidance resulting in better throughput.

References

1. C. Perkins, "RFC2002: IP mobility support," Oct. 1996.
2. C. Perkins, "IETF draft: Route optimization in mobile IP," Dec. 1997.
3. V. Jacobson, "Congestion avoidance and control," *Computer Communication Review*, vol. 18, no. 4, pp. 314-329, 1988.
4. D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *Computer Communication Review*, vol. 26, pp. 5-18, Apr. 1996.
5. D. S. Alexander, M. Shaw, S. M. Nettles, and J. M. Smith, "Active bridging," in *Proceedings of SIGCOMM '97*, 1997.
6. D. J. Wetherall, J. Guttag, and D. L. Tennenhouse, "ANTS: A toolkit for building and dynamically deploying network protocols," in *IEEE OPENARCH'98*, (San Francisco), Apr. 1998.
7. H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proceedings of the first Annual International Conference on Mobile Computing and Networking (Mobicom '95)*, pp. 2-11, Nov. 1995.

Adaptive Scheduling at Mobiles for Wireless Networks with Multiple Priority Traffic and Multiple Transmission Channels

Satish Damodaran¹ and Krishna M. Sivalingam^{2*}

¹ Cisco Systems, San Jose, CA 95134

² School of EECS, Washington State Univ., Pullman, WA 99164-2752

dsatish@cisco.com, krishna@eeecs.wsu.edu

Abstract. This paper considers scheduling algorithms for an infrastructure network where: (i) multiple channels are available per cell and (ii) mobile reservation and basestation scheduling protocol is used for medium access. The paper describes adaptive scheduling algorithms implemented at the mobile which modify the allocation provided by the base station. The motivation to do this is that under certain conditions, a mobile may wish to re-allocate its transmission slots between its priority queues based on the current queue status and other information. We propose an aging priority scheme where sessions who were forced to give up their allocated slots are provided increasing priority. The overall performance improvement for higher priority packets and corresponding performance degradation for lower priority packets is studied using discrete event simulation.

1 Introduction

Wireless Local Area Networks (WLANs) have become an increasingly important technology for today's computer and communications industry. In this paper, we consider an infrastructure based wireless network, where base stations are entities connected to the wired infrastructure; and mobiles communicate with the base station over the wireless link.

This paper studies channel sharing among the mobiles using a reservation and scheduling protocol as in [1]. Each user generates different types of sessions where each session traffic is associated with a priority. The paper also considers the situation where *multiple transmission channels* are available in a single cell. In [2], scheduling algorithms that run at the basestation for such a network were studied.

Those algorithms can be classified as *fully gated*, where packets arriving during the reservation interval must also wait for an entire cycle. This wait may not be desirable in certain situations. For example, a real time packet may miss a deadline, or when a mobile may need to transmit more higher priority packets quickly after recovering from an error state or due to battery power levels. Thus, the paper proposes a mechanism where the mobile modifies its transmission schedule provided by the base station. This is done by sending certain higher

* Corresponding Author. Part of the research was supported by Air Force Office of Scientific Research grants F-49620-97-1-0471 and F-49620-99-1-0125.

priority packets that arrive after the current reservation cycle in place of lower priority packets. This approach will definitely benefit the higher priority packets, but our design goal is to minimize the impact of performance on the lower priority packets.

The proposed algorithm implements an aging scheme which moves lower priority packets to higher priority queues based on the bandwidth consumed by each queue. To guarantee fairness, these sessions are later supplemented with additional bandwidth when the higher priority sessions have consumed too much of the bandwidth. The analysis is done using discrete event simulation methods.

2 Access Protocol and Scheduling

This section describes the access protocol framework for the multiple-channel system. The scheduling and queuing algorithms are typically used along with a medium access control (MAC) protocol.

Access Protocol: The access protocol considered in this paper is based on the EC-MAC protocol [1]. The access protocol is defined for an infrastructure network with a single base station serving mobiles in its coverage area. We are considering the protocol framework extended to multiple channels. Each mobile is equipped with a tunable transmitter and tunable receiver that can tune across multiple frequencies. The base station is assumed to have a set of transceivers – one per channel.

Transmission in EC-MAC protocol is organized by the base station into frames. Each frame has a fixed number of slots, where each slot equals the basic unit of wireless data transmission. The frame is divided into multiple phases. In the first phase, a frame synchronization message (FSM) is transmitted on the downlink and it contains framing, synchronization and other control information. The request/update phase comprises the uplink request transmissions from the mobiles to the base station and operates in a collision-less mode. New mobiles that just entered the area covered by the base station register themselves with the base station in the New-User phase. The schedule is then computed by the base station during the schedule computation phase and the computed schedule is broadcast to all the mobiles. The data phase is composed of both uplink and downlink traffic.

Overview of the Scheduling Algorithm: The uplink requests made by the M mobiles to the base station and the downlink traffic from the base station to the M mobiles over the C channels are represented by a $M \times 2P$ matrix, where P represents the number of priority queues within a mobile. Columns $1, \dots, P$ in the traffic vector indicate the slot requests from the M mobiles to the base station (uplink traffic), where each column represents a priority queue. Columns $P + 1, \dots, 2P$ in the traffic vector indicates the slot requests for the M mobiles from the base station. Thus, we only consider the first P columns of the matrix. We need to find a conflict-free assignment of the C channels such that the frame length is minimized. The rest of this section considers only uplink transmissions (from mobile to BS) with similar results applicable to the downlink.

Let d_{ij} denote the request made by mobile i for priority j traffic. Let $S = (S_i)$ denote the allocation vector, and let S_i denote the total amount of slots allocated on channel i where $0 \leq S_i, i = 1, \dots, C$.

The scheduling algorithm allocates the requests $d_{i1}, (i = 1, 2, \dots, M)$, contiguously (without any splitting of the requests) on the channel which currently has the least partial sum of allocation. If more than one channel has the same least partial sum, one channel is chosen at random for the next allocation. Once this is done, it computes the current schedule length, $\mathcal{L} = \max_{i=1}^C S_i$. Then it initializes $S_i = \mathcal{L}, (i = 1, 2, \dots, C)$, and allocates the requests of the next priority.

3 Adaptive scheduling

The previous section described the scheduling algorithm implemented at the BS. The main idea of this paper is to allow the mobiles to adapt this schedule based on local queue status and other conditions. The MAC at the mobile should implement some form of priority scheduling to select which of the queued packets get selected for transmission. Thus, the mobile chooses to send a higher priority packet instead of a lower priority packet, even though the latter packet was scheduled by the base station. This deferment may result in unfair treatment of lower priority packets leading to starvation. To overcome the unfairness, sessions suffering deferment in packet transmission must be supplemented with additional bandwidth later.

Adaptive scheduling using an aging scheme (ASAS): In this scheme, the priority of packets that are deferred is increased after each such deferment. This ensures that a packet will not starve by staying in a lower priority queue for ever. Consider an example where there are three sessions with respective priorities 0, 1, and 2 sharing a given link. At time i the mobile is scheduled by the base station to transmit packets belonging to queue 2. However there are packets waiting to be transmitted from higher priority queue 0 that arrived after the last reservation cycle. These packets are transmitted and the packets that were marked with priority 2 are now marked with priority 1. At a later point of time j ($j > i$), when priority 1 queue is chosen, these packets are transmitted.

A formal description of the algorithm is given in Figure 1. Procedure SelectQueue returns the next session from which a packet should be transmitted.

4 Performance Analysis

The following sections describe the source traffic models, performance metrics studied, and simulation results. The performance of the protocol has been studied through discrete-event simulation models written in C with YACSIM [3].

The simulation results presented here consider data traffic type for all priority sessions. In simulation, each mobile terminal data traffic is modeled as self-similar traffic with Hurst parameter of 0.9 [4]. The MAC parameters are: channel rate of 10 Mbps, TDMA frame length of 16 ms; number of slots per TDMA frame is 312 slots, number of slots in data phase is 254 slots, and data slot size is 64 bytes.

```

Algorithm SelectQueue(nodeid, slotOwner)
  If (slotOwner == 0) return (0);
  EndIf
  i = nodeid; j = slotOwner;
  For (k = 0, 1, 2, . . . , j − 1)
    If no packets in (sessionk) continue; EndIf
    If (ptik ≥ pmaxik) continue; EndIf
    (ptik)++;
    Move packet at the head of queue j to queue j − 1;
    return (k);
  EndFor
  return (j);
End.

```

Fig. 1. Procedure invoked to select the next session from which a packet is to be transmitted.

The performance metric studied is the average packet delay is defined as *the time between packet generation and packet transmission*. Another measure, the probability of meeting deadlines, is under investigation.

The discussions below are based on Fig. 2. We present the results for a system with 8 channels. There is not much variation in the average delay experienced by a priority 0 packet if few mobiles contend in the system. When the number of mobiles within a cell becomes more than 40, we see that packet delay tends to reduce with adaptive scheduling. We next consider average packet delay for priority 1 sessions. Beyond 30 mobiles, ASAS is able to lower the average packet delay experienced by priority 1 packets. However, beyond 80 mobiles, the delay experienced by these packets are higher. For priority 2 packets, the delay is actually less when the adaptive scheduling is used for systems which have between 25 and 50 mobiles. Beyond 50 mobiles, the delay of priority 2 packets is higher.

In summary, we find that ASAS is able to deliver better delay for networks with an intermediate number of nodes. For large number of nodes, the lower priority packets suffer performance degradation at the expense of higher priority. In [5], an alternate scheduling mechanism based on a fairness server is presented and analyzed.

5 Conclusion

This paper describes adaptive scheduling algorithms for a reservation-based MAC protocol for scheduling in wireless networks. An aging based adaptive scheduling algorithm for use at the mobiles was presented. Simulation studies that measured average packet delay were conducted. For high priority sessions, the packet delays were reduced considerably as required when the adaptive algorithms are used at the mobiles. The ASAS scheme does not perform as well for lower priority sessions for large number of nodes. Further work is necessary to limit the performance degradation of lower priority sessions.

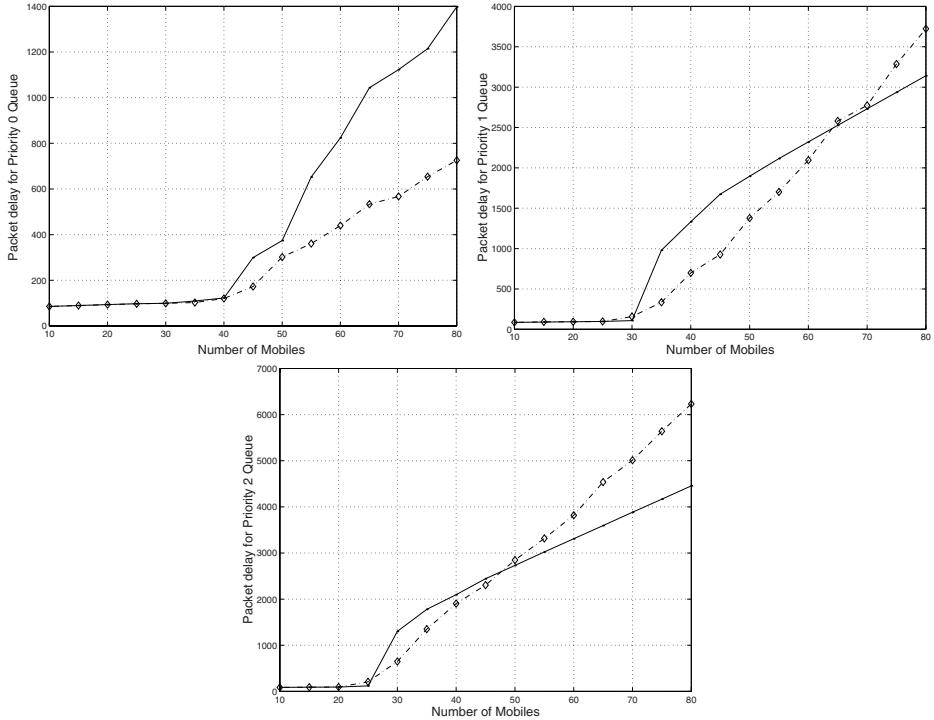


Fig. 2. Average Packet Delay for the three priorities. Solid and dotted lines represent the basic scheme (NAS) and aging based adaptive scheme (ASAS).

References

1. Krishna M. Sivalingam, Jyh-Cheng Chen, Prathima Agrawal, and Mani Srivastava, "Design and Analysis of Low-Power Access Protocols for Wireless and Mobile ATM Networks", *ACM/Baltzer Mobile Networks and Applications*, 1998, (Accepted for Publication).
2. Satish Damodaran and Krishna M. Sivalingam, "Scheduling in wireless networks with multiple transmission channels", in *International Conference on Network Protocols*, Toronto, Canada, Oct. 1999.
3. J. Robert Jump, *YACSIM Reference Manual*, Rice University, Department of Electrical and Computer Engineering, 2.1 edition, Mar. 1993.
4. Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson, "Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level", *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 71–86, Feb. 1997.
5. Satish Damodaran and Krishna Sivalingam, "Two different algorithms for adaptive scheduling at mobiles in multiple channel wireless local networks", Tech. Rep. TR99-DAWN-2, Washington State University, Email: krishna@eecs.wsu.edu, July 1999.

An Analysis of Routing Techniques for Mobile and Ad Hoc Networks

R. V. Boppana, M. K. Marina and S. P. Konduru

CS Division, The University of Texas at San Antonio

San Antonio, TX 78249-0667

{boppana,mmarina,skonduru}@cs.utsa.edu

Abstract. In this paper, we compare the performance of two on-demand and one pro-active algorithms for multi-hop, ad hoc networks and variants of the algorithms. We show that on-demand algorithms react well to transient conditions such as high node mobility in combination with low network load, but are unlikely to provide the best performance under heavy network loads.

1 Introduction

A mobile and ad hoc network (MANET) facilitates mobile hosts such as laptops with wireless radio networks communicate among themselves even when there is no wired network infrastructure. In a MANET, most hosts, if not all, are assumed to be moving continually and thus do not have a default router or fixed set of neighbors. So each mobile host should have an Internet Protocol (IP) routing algorithm for building and maintaining routing tables, just like an Internet router node. In this paper we analyze some of the most promising IP routing algorithms proposed for MANETs in recent literature for possible sources of overheads and optimizations. MANETs are characterized by shared wireless links or channels, which have low bandwidth and are unreliable owing to external noise and relative movement of nodes. Each node runs a medium access control protocol based on the IEEE 802.11 MAC standard.

The routing algorithms can be classified into pro-active and reactive protocols. The traditional IP routing algorithms such as distance vector (RIP [9]) and link-state (OSPF [11]) come under the category of pro-active algorithms. They build and maintain routing information about all the nodes in a network regardless of the usage or changes in the routes which appears to be wasteful in the context of ad hoc networks. Since the channel bandwidth is at a premium in these networks, many researchers proposed on-demand routing algorithms [14, 10, 12] which maintain only routes that are needed to send data packets currently in the network.

In this paper, we investigate the performance of the ad hoc on-demand distance vector (AODV) and the dynamic source routing (DSR) algorithms [14, 10] and compare them with the destination-sequenced distance vector (DSDV) proposed by Perkins and Bhagwat [13]. We also propose an adaptive version of DSDV algorithm and analyze its performance. While our analysis confirms some of the previously seen claims [2, 6] for AODV and DSR, it shows weaknesses of the two algorithms under a variety of traffic and mobility conditions. Based on this analysis, we describe features that are likely to work well and those that cause high overhead.

The rest of the paper is organized as follows. Section 2 will describe the basics and unique features of the algorithms considered in this study. Section 3 provides an analysis of the algorithms. Section 4 concludes the paper.

2 Routing Algorithms

We consider the original and a variant of DSDV (pro-active), and AODV and DSR (on-demand) algorithms.

2.1 Destination-sequenced distance vector (DSDV)

In the DV algorithms, each node maintains a table of routing entries, with each entry indicating destination node address, hop count and the next hop to reach the destination. Nodes broadcast their routing tables periodically to their neighbors, and incorporate any shorter routes obtained from neighbors. The major problem with the DV based algorithms is propagation of fallacious routing information among nodes, which leads to loops in routing paths. There are several solutions proposed to avoid this problem [3, 8, 13].

DSDV [13] solves the looping problem by attaching sequence numbers to routing entries. A node includes a next higher even sequence number for itself in its periodic and triggered updates. Any node that invalidates its entry to a destination, because of link failure, will increment the sequence number (which becomes an odd sequence number) and uses the new sequence number in its subsequent updates. Neighboring nodes having smaller sequence number for that destination than in the received update modify/invalidate their entry. An invalidated entry becomes valid only by the routing information propagated by the destination node with a higher even sequence number.

The triggered updates in DSDV consume a lot of channel bandwidth and, worse, they invalidate too many routing entries needlessly. We modify DSDV to limit the impact of triggered updates. Now, a node invalidates its routing entry based on a neighbor's update only if the neighbor's entry has a higher and odd sequence number and the neighbor is currently the specified next hop. A routing entry may be modified based on that of neighbor's if the neighbor has the higher even sequence number or better metric with the same sequence number. Also a node performs a triggered update only if more than a specified number of packets are queued up at a node. This reduces the routing overhead substantially for high traffic loads and high node mobility cases. We call the modified algorithm the adaptive destination-sequenced distance vector (ADSDV) algorithm.

2.2 Ad hoc on-demand distance vector (AODV)

AODV builds and maintains routing entries containing hop count, destination sequence number, and the next hop. Unlike DSDV, however, it does not use periodic or triggered updates to disseminate routing information. When a node needs to send a packet to a destination for which it has an invalid routing entry, a route is obtained by flooding the network with route request (RREQ) packets and obtaining responses (RREP packets) from nodes with routes to the destination.

An existing route entry may be invalidated if it is unused within the specified time interval (active route timeout period) or the next hop node is no longer a viable node to reach the destination. Invalidated routes are propagated to upstream neighbors that have used this node as their next hop. AODV requires the neighbors to exchange hello messages periodically or feedback from the link layer to detect the loss of a neighbor.

2.3 Dynamic source routing (DSR)

A routing entry in DSR contains all the intermediate nodes to be visited by a packet rather than just the next hop information as in DSDV and AODV. A source puts the entire routing path in the data packet, and the packet is sent through the intermediate nodes specified in the path (similar to the IP strict source routing option [5]). If the source does not have a route to the destination, it performs a route discovery as in the case of AODV, except that RREQ packets contain complete path information known up to that point and the RREP packets have the entire source route put by the destination.

DSR [10, 4] makes use of several optimizations to improve its performance. One of them is that the nodes can operate their interfaces in promiscuous mode and snoop on the source routes in the data packets and unicast routing packets. Also, an intermediate node may correct the path specified in a packet header if that path breaks. Snooping, which requires the device to be on all the time, may not be a feasible optimization for low-powered devices. To evaluate the effect of this optimization, we simulated two variants of DSR. In one variation, no snooping is allowed (non-snooping DSR or NSDSR) and in another a node is allowed to snoop only when it is doing route discovery (selective snooping DSR or SSDSR).

3 Performance Analysis

We used the *ns-2* network simulator [7] with the CMU extensions by Johnson et al. [4] for our analysis. The CMU extensions include implementations of DSDV, AODV, and DSR. The various parameters, timeouts and threshold values and optimizations used, unless explicitly specified, are exactly as described in the paper by Broch et al. [2]. For the sake of completeness, we outline the most relevant parameters for these algorithms.

For DSDV, the periodic update interval is 15 seconds. A node assumes a neighbor is lost if it does not hear the neighbor's update in 45 seconds (3*periodic update interval). For the proposed ADSDV, the periodic update interval is 5 seconds. Link layer feedback (provided by 802.11 LANs) is used to determine lost neighbors.

For AODV, we used the CMU implementation with one modification: the active route timeout is changed from 300 seconds (AODV300) to 3 seconds (AODV3), which is recommended by the latest specification of AODV [15].

In addition to the CMU implementation of DSR, we simulated the non-snooping DSR (NSDSR) and selective snooping DSR (SSDSR), which differ from the original DSR only in their snooping capabilities.

Network and mobility model. We simulated a network of 50 nodes randomly placed on a field of 1500m x 300m at the beginning of the simulation. Each node moves continuously, in a randomly chosen direction each time, at an average speed of 1 m/s, uniformly varied between 0-2 ms/, or 10 m/s, varied between 0-20 m/s [2].

A wireless channel based on 802.11 wireless LAN has 2 Mb/s bandwidth and a circular radio range with 250 meters radius. All protocols except for the original DSDV use the link-layer feedback to speedup detection of link breakages.

We used constant bit rate (CBR) traffic with 20 connections and 40 connections in our simulations. In each connection, the source sends 64-byte data packets at an average rate of 0.25-8 packets/second. Each simulation is started cold and the total simulation period is 1000 seconds.

Performance metrics. We use the average data packet latency and throughput (total number of data bits delivered) in Kb/s. To study the overheads of various routing algorithms, we plot routing packets transmitted/second and overhead bits/second. The overhead bits/s gives the bits transmitted as routing packets and source routing information (for DSR). All the metrics are plotted with respect to offered (data) load in Kb/s.

Performance of DSDV algorithms. Our simulations of DSDV and ADSDV indicate that ADSDV gives much improved latency and overhead than DSDV when the network load is high. ADSDV has more predictable routing overhead compared to DSDV, especially for the high node mobility case. Also the overhead in DSDV shoots up even at moderate loads, which explains why it has difficulty in sustaining throughput at high network loads. See [1] for more details.

Performance of DSR algorithms. Figure 1 gives the latencies and routing overheads for the original DSR algorithm with all the optimizations turned on and the non-snooping and selective snooping versions of DSR. The latencies vary very widely with traffic load. Essentially DSR and its variants use a lot of optimizations which lead to unpredictable behavior under transient conditions. At low traffic loads, data packets arrive at nodes infrequently, and most of the optimizations are done using stale routing information. Purging stale routes frequently will make DSR's performance more predictable. At moderate loads however, a clear trend can be seen. SSDSR and NSDSR perform substantially worse than the original DSR for high node mobility and moderate traffic loads. This difference in performance increases with higher traffic load. The overheads are higher when snooping is restricted or prohibited. A noteworthy point is selective snooping is only slightly better than not snooping at all. Since at most 40% of nodes send packets and thus can snoop while they do route discoveries, selective snooping is not very effective. If more nodes send packets, selective snooping can provide performance closer to that of DSR. Also, snooping is counterproductive at low traffic loads, since a lot of routing information becomes stale and causes more harm than good.

Comparison of ADSDV, AODV3, and DSR. To see how the best ones from each group of algorithms compare with one another, we did additional simu-

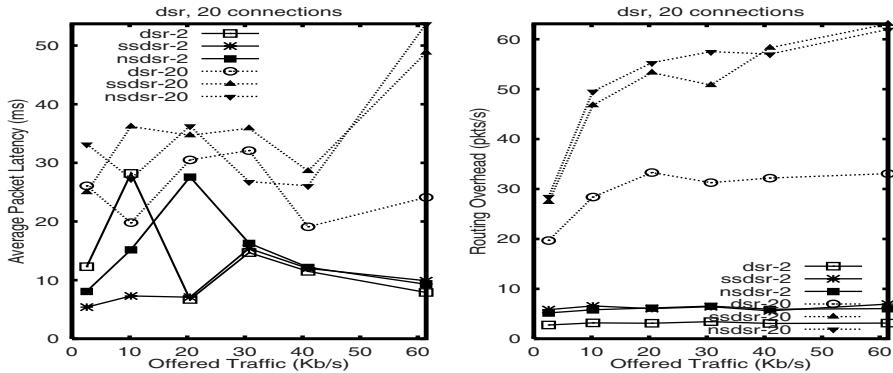


Fig. 1. Data packet latencies and overheads of the original and variants of DSR, algorithms for various traffic loads at low and high speeds. The number after the hyphen indicates the top speed by the mobile nodes.

lations with 40 CBR connections and data rates ranging from 1-8 packets/second (or approximately 20-160 Kb/s).

We describe here only the results for the high mobility case. (Results for the low mobility case are given in [1]). ADSDV outperforms both DSR and AODV by a wider margin. While AODV has lower latency for moderate network load, ADSDV is clearly the more stable algorithm for high network traffic (see Figure 2). ADSDV provides 35-43% higher throughput compared to both AODV and DSR. Furthermore, the routing overhead (Figure 2) is much less in packets/second. ADSDV has higher overhead in bits/sec, since its routing updates are much larger than route requests (RREQs) and route replies (RREPs) used in AODV and DSR.

4 Conclusions

The adaptive form of DSDV (ADSDV) presented is still a pro-active algorithm, since it depends mainly on periodic updates and controlled triggered updates, and seems to have superior performance to DSDV for increasing load in both low and high node mobility cases.

DSR incorporates too many optimizations. One of them, snooping data packets seems to be counter productive under transient network conditions (low load and high mobility). Overall, DSR is more stable and has lower overhead than AODV. AODV has better latencies than DSR and ADSDV for low traffic load, but has higher overhead when node mobility is high.

Compared to AODV and DSR, ADSDV performs poorly when the network load is low and node mobility is high. The main reason is the on-demand algorithms use route discovery to quickly learn paths, while ADSDV builds routes over a period of time. It seems preferable to use some form of route discovery with ADSDV to improve its performance at low loads. When the network

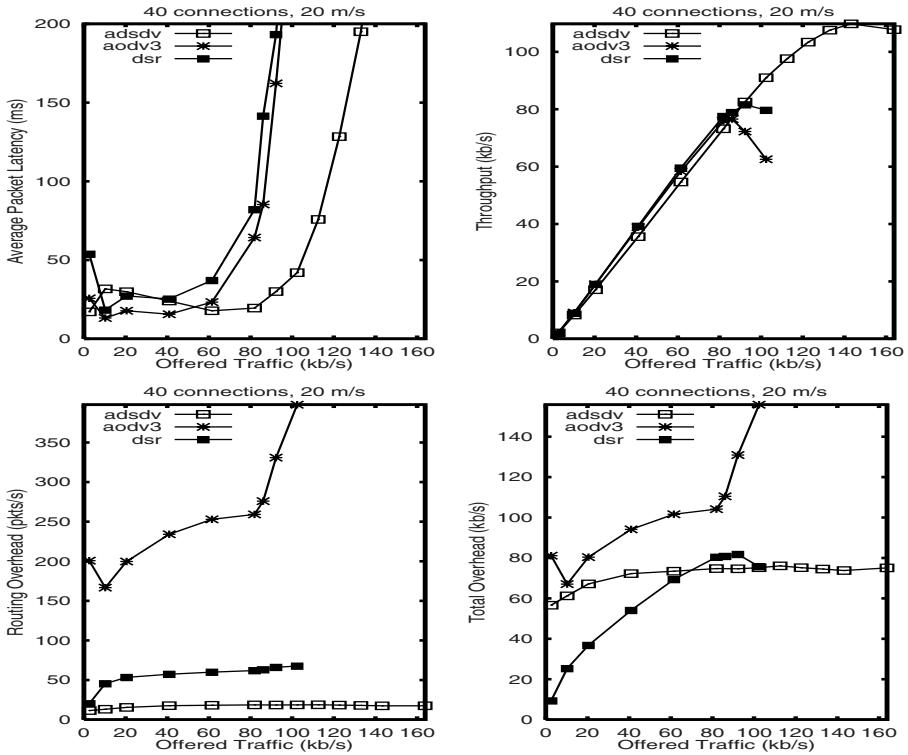


Fig. 2. Data packet latencies, throughputs, and overheads of ADSDV, AODV3 and DSR for the high node mobility case.

is really stressed, ADSDV outperforms both AODV and DSR. Furthermore, it exhibits graceful degradation of performance when the network is overloaded (beyond the point of saturation). This seems to suggest that carefully designed pro-active algorithms may be suitable or even preferable than pure on-demand techniques for routing in MANETs.

In future, we plan to augment ADSDV with route discovery and compare it with DSR, AODV and other routing algorithms.

Acknowledgments

This research has been partially supported by DOD/AFOSR grant F49620-96-1-0472 and NSF grant CDA 9633299. Marina and Konduru have also been supported by graduate fellowships from the University of Texas at San Antonio. The authors are grateful to the UC Berkeley/VINT project and the Monarch group at the Carnegie Mellon University for providing the simulator used in this study. The authors would like to thank Samir Das for many stimulating discussions during the course of this work.

References

1. R. V. Boppana, M. K. Marina, and S. P. Konduru, "An analysis of routing techniques for mobile and ad hoc networks." Manuscript <http://www.cs.utsa.edu/faculty/boppana/papers/papers.htm>, June 1999.
2. J. Broch et al., "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *ACM Mobicom '98*, Oct. 1998.
3. C. Cheng, R. Riley, and S. P. R. Kumar, "A loop-free extended Bellman-Ford routing protocol without bouncing effect," in *ACM SIGCOMM '89*, pp. 224–236, 1989.
4. CMU Monarch Group, "CMU Monarch extensions to the NS-2 simulator." Available from <http://monarch.cs.cmu.edu/cmu-ns.html>, 1998.
5. D. E. Comer, ed., *Internetworking with TCP/IP: volume 1*. Prentice Hall, Inc., 1995.
6. S. R. Das, R. Castaneda, and J. Yan, "Simulation based performance evaluation of mobile, ad hoc network routing protocols," in *Seventh Int'l Conf. on Computer Communication and Networks*, Oct. 1998.
7. K. Fall and K. Varadhan, "NS notes and documentation." The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC. Available from <http://www-mash.cs.berkeley.edu/ns>, Nov. 1997.
8. J. J. Garcia-Luna-Aceves, "A unified approach to loop-free routing using distance vectors or link states," in *ACM SIGCOMM '89*, pp. 212–223, 1989.
9. C. Hedrick, "Routing information protocol." RFC 1058, 1988.
10. D. B. Johnson et al., "The dynamic source routing protocol for mobile adhoc networks." IETF Internet Draft. <http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-02.txt>, 1999.
11. J. Moy, "Ospf version 2." RFC 1247, July 1991.
12. V. D. Park and S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *IEEE INFOCOM '97*, pp. 1405–1413, Apr. 1997.
13. C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance vector (DSDV) for mobile computers," in *ACM SIGCOMM '94*, pp. 234–244, Aug. 1994.
14. C. E. Perkins, "Ad hoc on demand distance vector routing." IETF Internet Draft. <http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-02.txt>, 1997.
15. C. E. Perkins, E. M. Royer, and S. R. Das, "Ad hoc on demand distance vector routing." IETF Internet Draft. <http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-03.txt>, 1999.

MobiDAT:Mobile Data Access and Transactions

Deepak Bansal, Manish Kalia, and Huzur Saran

Department of Computer Science and Engineering,
Block 6, Indian Institute of Technology,
New Delhi - 110016, India
saran@cse.iitd.ernet.in

Abstract. In this paper, we propose and study an application layer architecture to provide reliable data access and transactions in the Mobile environment. In our architecture, we use proxy agents at the base station and the mobile station to isolates the wireless channel errors from the mobile client as well as the wired network. To provide a reliable message exchange mechanism between the mobile and base station we introduce the concept of Mailbox. Using the proxy architecture, we propose new transaction mechanisms which provide transaction consistency and time bounds. We introduce the concept of Migrating Transaction Objects(MTO) and the Transaction Interface(TI). In these mechanisms we migrate a transaction from the mobile station to special proxies sitting on the base station in a reliable and secure manner. We have implemented the MobiDAT in an actual wireless framework using WaveLAN PCMCIA and ISA cards. We provide a performance evaluation of our architecture.

1 Introduction

The advancement in cellular technology and mobile communication have made it possible for a person to access information and data while on the move. Mobile computing presents new problems for reliable data (or information) access and transactions due to the intermittent and unreliable connectivity of the mobile with the network. More detailed explanation about the issues in mobile communication are discussed in [2]. Many architectures for the mobile environment have been proposed in literature [1]. However, none of these architectures discusses all the issues like disconnection recovery and connection state management in mobile data access and transactions. Carrying out transactions in the mobile environment poses several challenges. We need to ensure transaction consistency and time bounds. Work has been done in this area (such as [3], [4]), but it doesn't address the problem completely in the mobile domain.

2 Architecture

Our architecture has the following main components: Proxies, Mailboxes and Connection Recovery (see Figure 1(a1)).

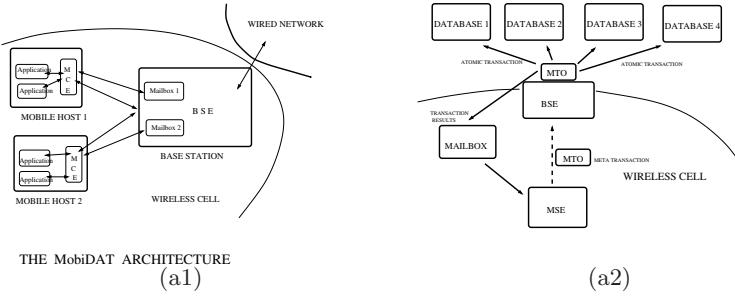


Fig. 1. (a1) The MobiDAT Architecture (a2) Transaction Mechanism in MobiDAT

A proxy is an application level entity which provides special services at the application layer. We use two proxies in MobiDAT, BSE (Base Station Entity) and MCE (Mobile Client Entity) residing respectively on the base station and mobile client. Applications on the mobile client communicate to the MCE data access and transaction requests. Data is fetched by the BSE on behalf of the mobile host and delivered to the mobile host via MCE. The wireless link is isolated from the applications by having a disconnection recovery protocol between the MCE and the BSE. To provide a reliable messaging protocol between the MCE and the BSE We propose the concept of Mailbox. Mailbox is a special storage area on the wired network reserved for an MCE. The MCE and BSE periodically monitor the mailbox for incoming messages and also places in the mailbox any outgoing messages. The mailbox also allows the MCE to store state recovery information thus enabling recovery from a power shutdown. A Mobile client does not have to remain always powered on to receive all incoming messages (thus saving power).

The restoration of communication involves firstly, re-establishment of connection and then, associating the new connection with the previous disrupted connection. One of the proxies re-initiates the broken connection. We introduce breakpoints in data being transferred. Thus, once a connection is disrupted, communication is recovered from the last breakpoint.

A transaction is defined as a sequence of data retrievals or updatations at a database. An atomic transaction refers to one such singular operation. We define a Meta transaction to be a collection of atomic transactions performing a particular task ex. in e-commerce buying or selling of a share is a meta transaction. Two important issues involved in transactions are consistency and time bounds for Meta transactions. A Meta transaction completes consistently if all the atomic transactions are completed. The current database packages support atomic transaction consistency. We focus on the consistency problem of Meta transactions initiated by a Mobile client with databases spatially distributed over the wired network. For the mobile user inconsistency can occur due to the frequent breakdown of the wireless link. Transactions have an associated time constraint of completion, beyond which they become invalid. An example is an

e-commerce transaction. To ensure consistency and give time bounds we shift the meta transaction to the wired network (BSE) using our proxy architecture (see Figure 1(a2)). The BSE conducts the transactions on behalf of the mobile client. The complete Meta transaction migrates as an MTO. The MTO needs reliable message exchange with the mobile. This is done using the mailbox. After the completion the results are communicated back (via the mailbox). To allow the migration of transactions and their execution at the BSE we needed to design a basic interface for the MTO. This allows BSE to be unaware of the transaction details (thus increasing security). This we call the transaction interface.

3 Results and Performance Evaluation

MobiDAT was implemented on an actual wireless Lan using AT&T Wavelan wireless cards. We introduced channel errors using the two state Markov chain error model [5]. This model comprises of a good state(0% drops) and a bad state(100% drops). There are two parameters in the model namely: α (the good state to bad state transition probability) and β (the bad state to bad state transition probability). Under normal conditions α and β were chosen to be 0.1 and 0.22 respectively and were varied to generate different error conditions.

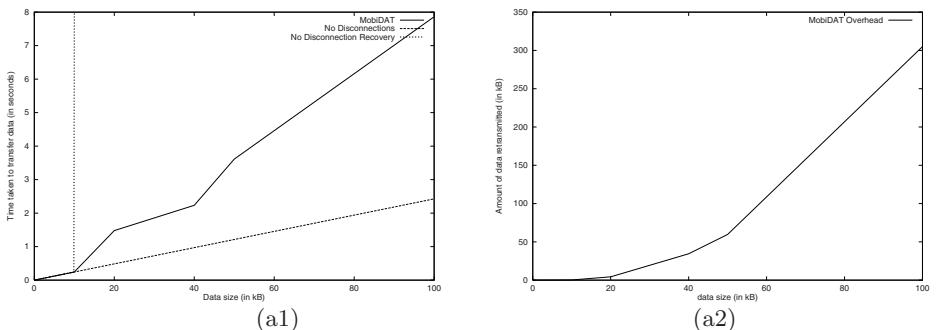


Fig. 2. (a1) Data transfer time in MobiDAT for average disconnection rate of 1 disconnection/10kB of data (a2) Amount of data retransmitted in connection recovery in MobiDAT for average disconnection rate 1 disconnection/10kB of data

First we evaluate the performance of disconnection recovery component of MobiDAT. Varying amount of data is accessed by the mobile user and various parameters are recorded. The disconnections occur at an average rate of 1 disconnection for 10 kB of data transferred. The results are presented in Figure 2. In Figure 2(a1), we plot the time taken for data transfer vs. the data size. In the absence of disconnections, time taken increases linearly with the data size. If disconnections occur and no recovery exists, an average of 10kB data gets

transferred successfully before the connection gets snapped. MobiDAT does disconnection recovery but this has an overhead in terms of the time taken to transfer the data. This overhead is due to disconnection discovery which has components from the system timers as they detect disconnections by timeouts. Overheads is also because of disconnection recovery and retransmission of data that did not reach the mobile host due to connection breaking. In Figure 2(a2), we plot the amount of data that needs to be retransmitted in connection recovery in MobiDAT vs the amount of actual data transferred. As the data size increases, the number of disconnections increases thus increasing retransmissions. We observe that MobiDAT scales well. In Figure 3(a1) and Figure 3(a2), we have plotted the time for data transfer and amount of data retransmitted for 100kB of data transfer vs disconnection rate. Here also, we see that as the disconnection rate increases the overhead in terms of time and retransmitted data increases linearly.

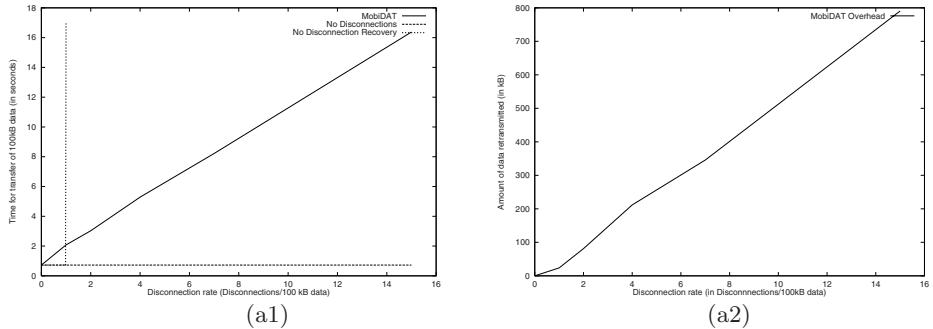


Fig. 3. (a1) Time for transfer of 100kB vs. Disconnection rate (a2)Retransmission for 100kB of data vs. Disconnection rate

We now present the performance of MobiDAT in ensuring the consistency and time bounds of transactions. We performed a sequence of meta transactions in both the unaided wireless case and in MobiDAT. We studied the transaction performance at different channel error conditions (different values of α and β). First the value of β was kept at 0.22 and the value of α was varied from 0 to 1. Then the value of α was kept to 0.1 and the value of β was varied. The number of transactions completed inconsistently was recorded. From Figure 4(a1) and Figure 4(a2) we observe, MobiDAT gives a lower average transaction time. As the errors increase the performance of unaided wireless deteriorates. For high values of α the performance of unaided wireless deteriorates more than in case of high β . This is because the link goes into bad state very frequently and transactions are disrupted. MobiDAT ensures consistent transactions. For unaided wireless the inconsistency increases with higher channel errors. For high values of β the inconsistency is large as there are prolonged periods of connectivity. If the

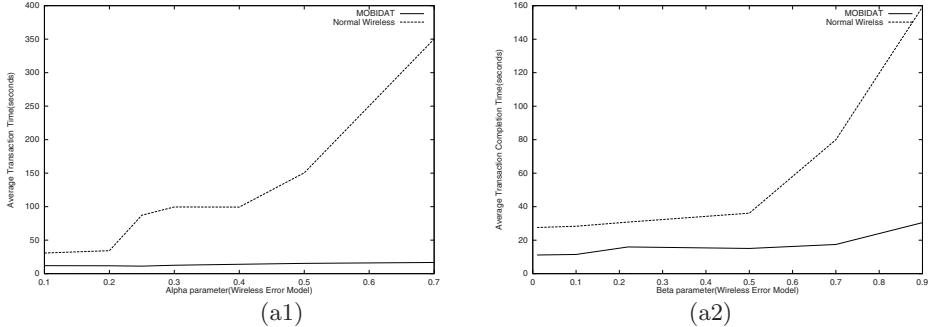


Fig. 4. (a1) Variation of Average Transaction Completion time vs. α factor(wireless error model) for $\beta = 0.22$ (a2)Variation of Average Transaction Completion time vs. β factor(wireless error model) for $\alpha = 0.1$

wireless channel error characteristics change frequently, a time bound cannot be given for unaided wireless. In MobiDAT the variation in the average transaction completion time is negligible and hence is bounded. Even in static channel errors MobiDAT gives lower bounds.

4 Conclusions

Through our implementation and performance evaluation we find that MobiDAT performs very well. MobiDAT enables large data access by alleviating the problems of disconnection in the wireless. Also, MobiDAT ensures reliable messaging by using the mailboxes. Using MobiDAT we were able to eliminate the problem of Meta Transaction Inconsistency and give time bounds.

References

1. George Samaras and Evangelia Pitoura, “Computational Models for Wireless and Mobile Environment”, *Technical Report 98-9, Computer Science Department, University of Ioannina, Greece*, 1998.
2. M. Satyanarayanan, “Fundamental Challenges in Mobile Computing”, *PODC 1996: 1-7*
3. J. Gray and A. Reuter, “Transaction Processing: Concepts and Techniques”, *Morgan Kaufman*, 1993.
4. O. Buhkres, S. Morton, P. Zhang, E. Vanderdijs, C. Crawley, M. Mossman, J. Platt, “A Proposed Mobile Architecture for Distributed Database Environment”, *5th Euromicro Workshop on Parallel and Distributed Processing*, 1997
5. Anurag Kumar, “Comparitive performance analysis of versions of TCP in a local network with a lossy link”, *IEEE ACM transactions on Networking*, Vol 6, No. 4, August 1998.

Session IV-A

Parallel Algorithms - II
Chair: Dilip Krishnaswamy
Intel Corporation

Optimal k -Ary Divide&Conquer Computations on Wormhole 2-D and 3-D Meshes

Jan Trdlička and Pavel Tvrdík*

Department of Computer Science and Engineering
Czech Technical University, Karlovo nám. 13
121 35 Prague, Czech Republic
`{trdlicka,tvrdik}@sun.felk.cvut.cz`

Abstract. In this paper, we describe optimal and asymptotically optimal embeddings of k -nomial trees into 2-D and 3-D meshes with wormhole routing. We consider both embeddings that ignore dimension-order routing and embeddings that map tree edges only to paths conforming to the dimension-order routing. The embeddings provide optimal algorithms for non-pipelined k -ary Divide&Conquer computations in wormhole meshes. Our results generalize what has been known for the binary case and they achieve trivial lower bounds for $k = 3$. For $4 \leq k \leq 7$, we give lower-bound matching solutions in 2-D meshes with arbitrary routing.

Keywords: Divide&Conquer computation, k -nomial tree, wormhole meshes.

1 Introduction

A Divide&Conquer (D&C) computation consists of a *divide* communication phase, a *computation* phase in leaf nodes, and a backward *combine* communication phase. A usual solution for many problems is the *binary* D&C, but not always. Sometimes, the problem does not require a specific divide factor k and choosing $k > 2$ may speedup the computation (greater k implies less levels). Or some algorithms lead naturally to k -ary D&C with $k \geq 3$. For example, Karatsuba multiplication [2] implies $k = 3$ and Strassen matrix multiplication [7] implies $k = 7$. This raises the question of efficient implementations of k -ary D&C computations. While binary D&C received considerable attention, the k -ary case is nearly untouched.

D&C computations exist in two flavors. In a *pipelined* D&C computation (PDC), the master process is the root of a k -ary tree T and keeps sending waves of data in a pipeline way into T . If $h(T)$ denotes the height of T , then up to $2h(T) + 1$ computational waves can coexist simultaneously in T . The latency of one step is determined by the slowest member of the pipeline, typically the leaf computation or the master divide steps. A k -ary h -level PDC can be efficiently

* This research has been supported by GAČR under grant #102/97/1055.

simulated on a network H by embedding the complete k -ary tree of height h , $CT_{k,h}$, into H with load 1. These embeddings are relatively well understood in the binary case ($k = 2$) if H is a 2-D or 3-D mesh [4,5]. These results have been recently generalized for any k by constructing embeddings of $CT_{k,h}$ into wormhole (WH) 2-D (3-D) meshes with load 1 and edge congestion twice (four times) the trivial lower bound [9,10].

The other type is the *non-pipelined* D&C computation (NDC). At one time, only one wave of computation is active in T . A simulation of a k -ary NDC on a network H by embedding $CT_{k,h}$ into H with load 1 is wasteful, since each H 's node simulates either an internal node or a leaf and is therefore underutilized. There are two ways to make the NDC computation efficient. The first one is to use an embedding that loads (nearly) every node of H with exactly one tree leaf and with at most one internal tree node. This was done nicely in [3] for $k = 2$ and for 2-D WH meshes. The other approach to NDC is called “keep half, send half” in the binary case. It guarantees that every node of H , once being activated by the divide wave, remains active in the communication phase and turns to a computational slave in the computation phase. The corresponding tree structure in case of $k = 2$ is called the *binomial tree*. The binomial tree of height h , BT_h , consists of two BT_{h-1} whose roots are joined with an edge. A network H can simulate an h -level binary NDC if there is an efficient embedding of BT_h into H with load 1. Two solutions for 2-D meshes were given in [6]: one with minimal dilation for store-and-forward meshes and one with minimal edge congestion for WH meshes. The latter embedding nicely conforms even to XY -routing, since due to the binary recursiveness of BT_h , each edge of BT_h is mapped either on a horizontal or vertical path in the mesh.

In this paper, we generalize these results to general k -ary NDC. We introduce the notion of k -nomial trees that correspond to the paradigm “keep the k -th part and send $k - 1$ remaining parts”. We describe asymptotically optimal k -ary NDCs in 2-D and 3-D WH meshes, that either ignore or assume dimension-order routing. All results are based on constructions of embeddings of k -nomial trees into meshes with load 1 and with congestion either matching the lower bound or at most twice greater than the lower bound. All embeddings take advantage of the hierarchical recursiveness of both k -nomial trees and meshes and the constructions are described inductively on the height h of the k -nomial tree with fixed k . Due to the lack of space, we describe the results for 2-D meshes only and the reader is referred to [8] for the results on 3-D meshes.

2 Basic Definitions

A graph G is a pair $(N(G), E(G))$, where $N(G)$ is the set of *nodes* of G and $E(G)$ is the set of *edges* of G . An edge joining nodes u and v is denoted by $\langle u, v \rangle$. The k -nomial tree $NT_{k,h}$ of height h is defined inductively.

- $NT_{k,0}$ is a single node with no edges.
- $NT_{k,h}$ consists of k copies of $NT_{k,h-1}$ and the root of one $NT_{k,h-1}$ is the parent of roots of the remaining $k - 1$ $NT_{k,h}$'s (see Figure 1).

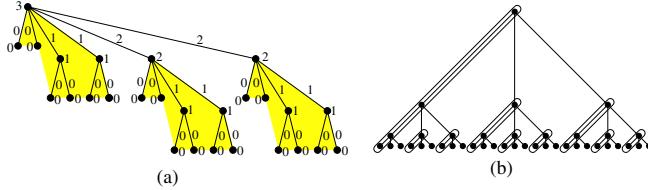


Fig. 1. (a) $NT_{3,3}$. Node and edge labels represent levels. (b) 3-nomial tree as a collapsed complete 3-ary tree.

Figure 1 shows that a k -nomial tree is isomorphic to a complete k -ary tree with chains of left-most children collapsed into a single node. We define the *level* $l(u)$ of node u in $NT_{k,h}$ as the *maximum* distance between u and its descendant leaf. The root of $NT_{k,h}$ is at level h and leaves at level 0. Similarly, the *level* of edge $e = \langle u, v \rangle$ is $l(e) = \min\{l(u), l(v)\}$. A node at level l is joined with its parent via an edge at level l and is incident to $l(k-1)$ -tuples of edges, where the i -th tuple consists of edges at level i , for $i = 0, \dots, l-1$. The edges used in the j -th communication step of an h -step NDC are exactly the edges at level $h-j$ in $NT_{k,h}$.

A 2-D mesh with a columns and b rows is denoted by $M_{a,b}$. Rows (columns) are numbered bottom up (left to right) starting from 1. Node in column x and row y is denoted by $[x, y]$. A rightward horizontal path in a mesh is called an X^+ path, a leftward one is an X^- path. Similarly for Y^+ , Y^- , $Y^-X^+Y^+$ paths, and so on. Similar definitions apply for 3-D meshes. By *dimension-order routing* in meshes, we mean XY -routing in 2-D meshes and XYZ -routing in 3-D ones.

An embedding of a *guest* graph G into a *host* graph H is a pair of mappings (ϱ_N, ϱ_E) where ϱ_N maps nodes $N(G)$ onto nodes $N(H)$ and ϱ_E maps edges $E(G)$ onto paths in H . The *load* of (ϱ_N, ϱ_E) is the maximum number of nodes in G mapped onto a node in H . The *expansion* of (ϱ_N, ϱ_E) , expn , is the ratio $|N(H)|/|N(G)|$. Since we will deal only with $NT_{k,h}$ as the guest graph G , we define *level- i congestion* under (ϱ_N, ϱ_E) , $\text{leng}(i)$, to be the maximum number of edges of level i from $NT_{k,h}$ mapped onto an edge of H . Note that this definition is similar to the *clocked congestion* in [1].

For the sake of brevity, we define $s = \lceil k/2 \rceil$ and $\lambda(i) = s + (s-1)k^{\frac{k^i-1}{k-1}-1}$.

3 k-Ary NDC Computations in WH 2-D Meshes

The first embedding does not take the dimension-routing into consideration.

Theorem 1. Let $k \geq 3$ and $h \geq 2$. Then $NT_{k,h}$ can be embedded into $M_{k^{\lceil h/2 \rceil}, k^{\lfloor h/2 \rfloor}}$ with $\text{expn} = 1$, $\text{load} = 1$, $\text{leng}(0) = \text{leng}(1) = \lceil (k-1)/2 \rceil$, and $\text{leng}(q) = \lceil (k-1)/4 \rceil$ for $q \geq 2$.

Proof. 1. $h = 2$. Figure 2 illustrates an embedding of $NT_{k,2}$ into $M_{k,k}$ with load = 1. The root of $NT_{k,2}$, depicted by the black circle, is mapped onto mesh

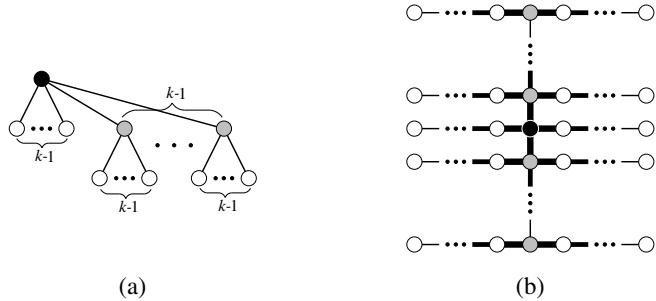


Fig. 2. (a) $NT_{k,2}$. (b) Embedding of $NT_{k,2}$ into $M_{k,k}$.

node $[s, s]$ and its $k - 1$ children (grey circles) are mapped onto the remaining nodes of column s . The leaves of $NT_{k,2}$ are mapped onto the remaining nodes of their parent's row. Clearly, $\text{expn} = 1$, $\text{load} = 1$, $\text{lcng}(0) = \text{lcng}(1) = \lceil (k - 1)/2 \rceil$. The induction step slightly differs for h odd and h even.

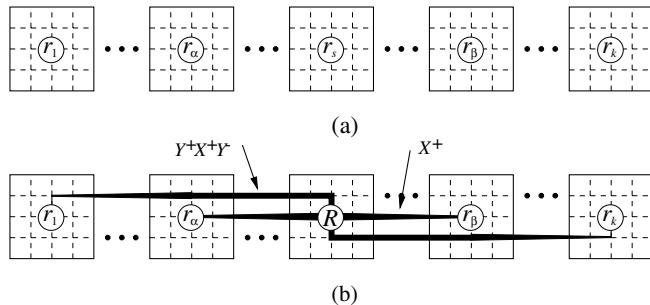


Fig. 3. Embedding of $NT_{k,2i+1}$ into M_{k^{i+1}, k^i} . (a) Subtrees $NT_{k,2i}$ embedded into submeshes M_{k^i, k^i} by induction. (b) Node r_s becomes the root R of $NT_{k,2i+1}$.

2. $h \geq 3$ is odd. Let $h = 2i + 1$, $i \geq 1$. Figure 3 illustrates an embedding of $NT_{k,h}$ into M_{k^{i+1}, k^i} . Along axis X , mesh M_{k^{i+1}, k^i} consists of k submeshes M_{k^i, k^i} , each with one copy of $NT_{k,h-1}$ embedded by induction. The root of the j -th $NT_{k,h-1}$ is denoted by r_j . Hence, $\varrho_N(r_j) = [(j - 1)k^i + \lambda(i), \lambda(i)]$ for $1 \leq j \leq k$, $\text{expn} = 1$, $\text{load} = 1$, $\text{lcng}(0) = \text{lcng}(1) = \lceil (k - 1)/2 \rceil$, and $\text{lcng}(q) = \lceil (k - 1)/4 \rceil$ for $q = 2, \dots, h - 2$. Node r_s then becomes the root R of $NT_{k,h}$ (see Figure 3(b)). Let $\alpha = \lceil (s - 1)/2 \rceil$ and $\beta = s + \lceil (k - s)/2 \rceil$. Then root R is linked to roots $r_1, \dots, r_{\alpha-1}$ by $Y^+X^-Y^-$ paths, to r_α, \dots, r_{s-1} by X^- paths, to r_{s+1}, \dots, r_β by X^+ paths, and to $r_{\beta+1}, \dots, r_k$ by $Y^-X^+Y^+$ paths. Therefore, $\text{lcng}(h - 1) = \lceil (k - 1)/4 \rceil$.

3. $h \geq 4$ is even. The construction is basically the same, except for interchanged directions X and Y . \square

This embedding is optimal and cannot be improved for $k = 3$. For $k \geq 4$, $\text{lcng}(0)$ and $\text{lcng}(1)$ is twice the trivial lower bound. Figure 4 shows optimal embeddings of $NT_{k,2}$ into $M_{k,k}$ for $4 \leq k \leq 7$. We conjecture that such embeddings with $\text{lcng}(0) = \text{lcng}(1) = \lceil (k-1)/4 \rceil$ exist for any $k \geq 4$, but we do not have a general construction. We can do it if we sacrifice the unit expansion. But if we insist on $\text{expn} = 1$, it seems to be a difficult problem. However, all practical k -ary NDC we know about use $k \leq 7$. Therefore, Theorem 1 together with constructions in Figure 4 provide the lower-bound matching solutions for all practical applications.

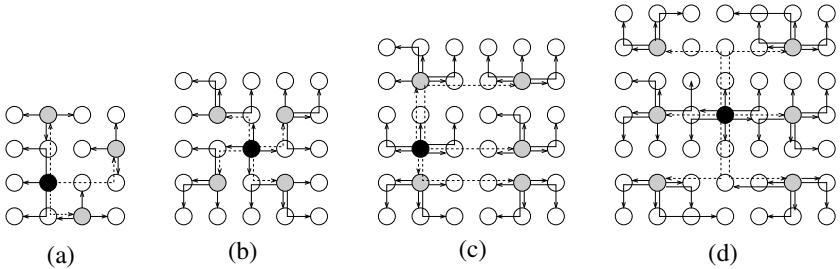


Fig. 4. Embedding of $NT_{4,k}$ into $M_{k,k}$ for $4 \leq k \leq 7$. (a) $k = 4$. (b) $k = 5$. (c) $k = 6$. (d) $k = 7$.

4 k -Ary NDC Computations in WH 2-D Meshes with XY-Routing

It turns out that dimension-order routing does not allow for the level congestion to match the lower bound, since in some steps, parents cannot use all four directions to communicate with their $k - 1$ children.

Theorem 2. *Let $k \geq 3$ and $h \geq 2$. Then $NT_{k,h}$ can be embedded into $M_{k^{\lceil h/2 \rceil}, k^{\lfloor h/2 \rfloor}}$ with $\text{expn} = 1$, $\text{load} = 1$, $\text{lcng}(0) \leq s$, $\text{lcng}(q) = \lceil (k-1)/2 \rceil$ for $q = 1, 2, 4, 6, \dots$, and $\text{lcng}(q) = \lceil (k-1)/3 \rceil$ for $q = 3, 5, 7, \dots$, so that the tree edges are mapped onto mesh paths conforming to XY-routing.*

Proof. 1. $h = 2$. To minimize the level congestion under the constraints of the dimension-order routing across induction steps, we need to use 2 slightly different embeddings of $NT_{k,2}$ into $M_{k,k}$, denoted by I and II. Embedding I is exactly the same as in Figure 2(b) and embedding II differs in the position of the root R of $NT_{k,2}$: it is mapped onto mesh node $[s+1, s]$ instead of $[s, s]$.

2. $h \geq 3$ is odd. Let $h = 2i + 1$, $i \geq 1$. Let us describe embedding I of $NT_{k,h}$ into M_{k^{i+1}, k^i} . M_{k^{i+1}, k^i} consists horizontally of k submeshes M_{k^i, k^i} , each with one copy of $NT_{k,h-1}$ embedded by induction (see Figure 5(a)).

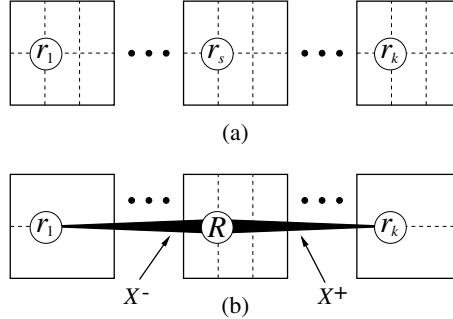


Fig. 5. Embedding I of $NT_{k,2i+1}$ into M_{k^{i+1},k^i} , $i \geq 1$. (a) Embedding I of k subtrees $NT_{k,2i}$ into k submeshes M_{k^i,k^i} by induction. (b) Node r_s becomes the root of $NT_{k,2i+1}$.

Therefore, $\varrho_N(r_j) = [(j-1)k^i + \lambda(i), \lambda(i)]$ for $1 \leq j \leq k$. expn = 1, load = 1, lcng(0) $\leq s$, lcng(q) = $\lceil (k-1)/2 \rceil$ for $q = 1, 2, 4, \dots, h-3$, and lcng(q) = $\lceil (k-1)/3 \rceil$ for $q = 3, 5, 7, \dots, h-2$. Node r_s becomes the root R of $NT_{k,h}$ (see Figure 5(b)). The dimension-order routing allows only X^+ and X^- paths for linking R to other roots r_j , $j = 1, \dots, s-1, s+1, \dots, k$, and therefore lcng($h-1$) = $\lceil (k-1)/2 \rceil$.

Embedding II of $NT_{k,h}$ into $M_{i+1,i}$ is the same, but the induction step uses embedding II, i.e., $\varrho_N(r_j) = [(j-1)k^i + \lambda(i) + 1, \lambda(i)]$ for $1 \leq j \leq k$.

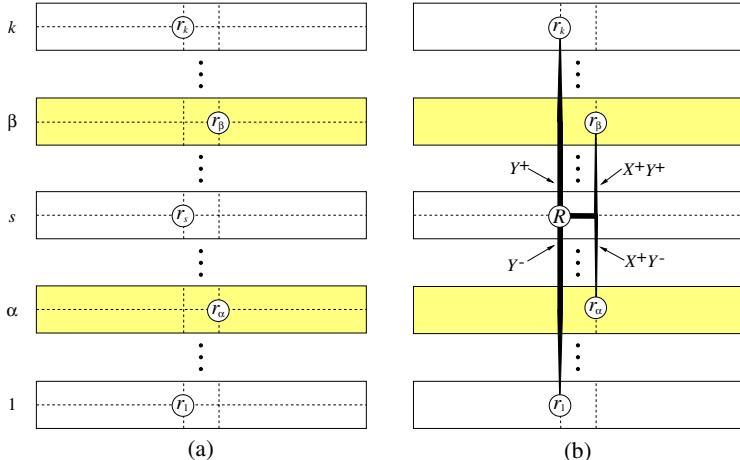


Fig. 6. Embedding I of $NT_{k,2i}$ into M_{k^i,k^i} if $i \geq 2$. (a) Embeddings I and II of k subtrees $NT_{k,2i-1}$ into k submeshes $M_{k^{i-1},k^{i-1}}$ by induction. (b) Node r_s becomes the root R of $NT_{k,2i}$.

3. $h \geq 4$ is even. Let $h = 2i$, $i \geq 2$. To preserve induction conditions, we again need two similar embeddings of $NT_{k,h}$ into M_{k^i,k^i} . Figure 6 describes embedding I. Let $\alpha = \lceil (k-1)/3 \rceil + 1$, $\beta = k - \lceil (k-1)/3 \rceil$, $A = \{1, \dots, \alpha-1, s, \beta+1, \dots, k\}$, and $B = \{\alpha, \dots, s-1, s+1, \dots, \beta\}$. Mesh M_{k^i,k^i} consists vertically of k submeshes $M_{k^i,k^{i-1}}$, indexed by numbers $1, \dots, k$. By induction, subtrees $NT_{k,h-1}$ are embedded by embedding I into submeshes with indexes in A and by embedding II into submeshes with indexes in B . Hence, $\varrho_N(r_j) = [\lambda(i), (j-1)k^{i-1} + \lambda(i-1)]$ if $j \in A$, $\varrho_N(r_j) = [\lambda(i)+1, (j-1)k^{i-1}\lambda(i-1)]$ if $j \in B$, $\text{expn} = 1$, $\text{load} = 1$, $\text{lcng}(0) \leq s$, $\text{lcng}(q) = \lceil (k-1)/2 \rceil$ for $q = 1, 2, 4, \dots, h-2$, and $\text{lcng}(q) = \lceil (k-1)/3 \rceil$ for $q = 3, 5, 7, \dots, h-3$. Node r_s again becomes the root R of $NT_{k,h}$. It is linked to other roots r_j using Y^+ , Y^- , X^+Y^+ , and X^+Y^- paths (see Figure 6(b)), which gives $\text{lcng}(h-1) = \lceil (k-1)/3 \rceil$.

The embedding II of $NT_{k,h}$ into M_{k^i,k^i} is *dual symmetric* to embedding I: embedding I (II) is applied by induction in submeshes with indexes in B (A), respectively. \square

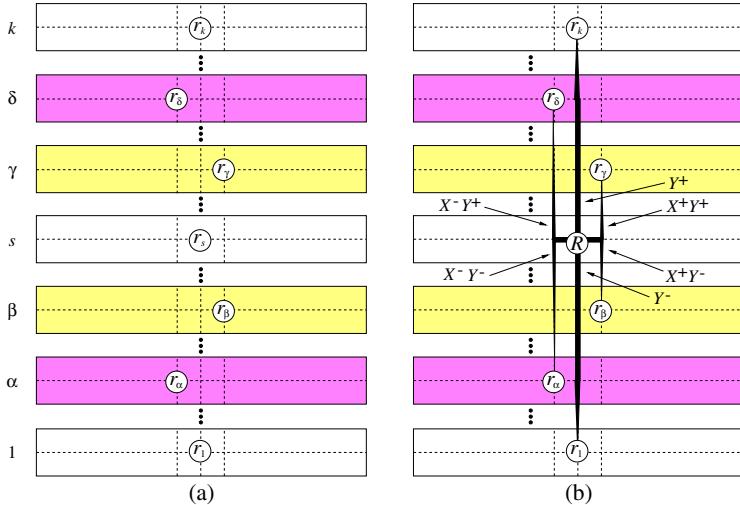


Fig. 7. Another embedding of $NT_{k,2i}$ into M_{k^i,k^i} .

Since we alternate directions X and Y in odd and even steps (and it is the only reasonable way to deal with recursivity of 2-D meshes), there is no way how to improve the congestion in odd steps. Can we do better in even steps? It seems unlikely in general. Figure 7 shows a possible better construction using all four directions. But if this scheme is applied recursively, the number of columns the roots of subtrees are mapped onto increases by 2 in every even step. Therefore, it can be used only if $h < k$. Otherwise, the roots of subtrees diffuse beyond boundaries of its native basic submesh $M_{k,k}$ and therefore the boundaries of

the whole mesh (or they bounce back if they hit boundary of their $M_{k,k}$) and vertical mesh edges will be multiple-congested.

5 k -Ary NDC Computations in WH 3-D Meshes

The results for 3-D meshes are more complicated, see [8] for details.

Theorem 3. *Let $k \geq 3$ and $h \geq 3$. If dimension-ordered routing is ignored, then $NT_{k,h}$ can be embedded into $M_{k^{\lfloor(h+2)/3\rfloor}, k^{\lfloor(h+1)/3\rfloor}, k^{\lfloor h/3\rfloor}}$ with $\text{expn} = 1$, load = 1, $\text{lcn}(0) = \text{lcn}(1) = \text{lcn}(2) = \lceil(k-1)/2\rceil$, and $\text{lcn}(q) = \lceil(k-1)/6\rceil$ for $q \geq 3$. If the tree edges are mapped onto mesh paths conforming to XYZ-routing, then $NT_{k,h}$ can be embedded into $M_{k^{\lfloor(h+2)/3\rfloor}, k^{\lfloor(h+1)/3\rfloor}, k^{\lfloor h/3\rfloor}}$ with $\text{expn} = 1$, load = 1, $\text{lcn}(0) = \lceil(k-1)/2\rceil$, $\text{lcn}(1) = \text{lcn}(2) \leq s$, $\text{lcn}(q) = \lceil(k-1)/2\rceil$ for $q = 3, 6, 9, \dots$, $\text{lcn}(q) = \lceil(k-1)/3\rceil$ for $q = 4, 7, 10, \dots$, and $\text{lcn}(q) = \lceil(k-1)/4\rceil$ for $q = 5, 8, 11, \dots$*

6 Conclusions and Open Problems

Many problems remain open. The most important one is whether there are good embeddings of k -nomial trees into arbitrary 2-D or 3-D meshes. But this is a hard problem even for $k = 2$.

References

1. T. Andreae et al. On embedding 2-dimensional toroidal grids into de Bruijn graphs with clocked congestion. Technical report, University of Hamburg, 1995.
2. G. Cesari and R. Maeder. Performance analysis of the parallel Karatsuba multiplication algorithm for distributed memory architectures. *Journal of Symbolic Computation*, 21(4/5/6):467–474, 1996.
3. A. Gibbons and M. Patterson. Dense edge-disjoint embedding of binary trees in the meshes. In *Proc. of the 4th Ann. ACM SPAA*, pages 257–263, 1992.
4. S. K. Lee and H. A. Choi. Embedding of complete binary trees into meshes with row-column routing. *IEEE T-PDS*, 7(5):493–497, May 1996.
5. S. K. Lee and H. A. Choi. Link-disjoint embedding of complete binary trees in meshes. *Networks*, 30:283–292, 1997.
6. V. Lo et al. Parallel divide and conquer on meshes. *IEEE T-PDS*, 7(10):1049–1057, October 1996.
7. V. Strassen. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In *7th Ann. Symp. FOCS*, pages 49–54. IEEE Computer Society Press, 1986.
8. J. Trdlicka. *Efficient embeddings of treelike parallel computations into orthogonal interconnection networks*. PhD thesis, CTU Prague, 1999.
9. J. Trdlička and P. Tvrdík. Embedding complete k -ary trees into 2-D meshes with optimal edge congestion. In Y. Pan et al., editors, *Proc. of the 10th IASTED Int. Conf. on Par. and Distr. Computing Systems*, pages 51–55. ACTA Press, 1998.
10. J. Trdlička and P. Tvrdík. Embedding complete k -ary trees into 3-D meshes with optimal edge congestion. In G. Gupta et al., editors, *Proc. of the 2nd IASTED Int. Conf. on Par. and Distr. Comp. and Networks*, pages 396–401. ACTA Press, 1998.

Parallel Real Root Isolation Using the Descartes Method *

Thomas Decker and Werner Krandick

Department of Mathematics and Computer Science,
University of Paderborn, Germany
`{Decker, Krandick}@upb.de`

Abstract. Two new scheduling algorithms are presented. They are used to isolate polynomial real roots on massively parallel systems. One algorithm schedules computations modeled by a pyramid DAG. This is a directed acyclic graph isomorphic to Pascal's triangle. Pyramid DAGs are scheduled so that the communication overhead is linear. The other algorithm schedules parallelizable independent tasks that have identical computing time functions in the number of processors. The two algorithms are combined to schedule a tree-search for polynomial real roots; the first algorithm schedules the computations associated with each node of the tree; the second algorithm schedules the nodes on each level of the tree.

1 Introduction

Real root isolation is the task of finding disjoint isolating intervals for all the real roots of a given polynomial; those are intervals that contain exactly one root each. In sequential computation, root isolation is performed efficiently by the (infallible) Descartes method [CA76,Kra95]. We present two new scheduling algorithms that can be used to obtain an efficient parallel version of the Descartes method.

One scheduling algorithm schedules computations modeled by a pyramid DAG. This is a directed acyclic graph (DAG) isomorphic to Pascal's triangle of some height. We analyze our algorithm in the *LogP*-model [CKP⁺93] and compare it to a method suggested by Kumar et al. [KK93] and to a method due to Lewandowski et al. [LCB96]. Our algorithm reduces the communication overhead from a quadratic term to a linear one and thus allows more scalability than the two other methods.

The other scheduling algorithm schedules parallelizable independent tasks that have identical computing time functions in the number of processors. The set of these tasks is partitioned into subsets whose elements are scheduled for parallel execution; the subsets themselves are scheduled for sequential execution. We give an algorithm for computing partitions that induce a minimum makespan; the computing time of the scheduling algorithm is dominated by the square of the number of tasks.

The two scheduling algorithms are used in our parallel version of the Descartes method to schedule a tree-search for polynomial real roots. The computations associated

* Supported by German Science Foundation (DFG) Project SFB-376, by European Union ESPRIT LTR Project 20244 (ALCOM-IT), and by European Union TMR-grant ERB-FMGE-CT95-0051.

with each node of the search tree are pyramid DAGs, and the first algorithm is used to schedule them. The set of pyramid DAG computations associated with a level of the search tree is a set of uniform parallelizable independent tasks, and the second algorithm is used to schedule them. If the search tree is narrow, our first scheduling method leads to a good efficiency. If the search tree is wide, the second scheduling method further reduces the computing time. By contrast, the parallel Descartes method proposed by Collins, Johnson, and Küchlin [CJK92] performs badly on narrow search trees since it computes the DAGs sequentially.

2 Scheduling Algorithms for the Pyramid DAG

A *pyramid DAG* is a directed acyclic graph that is isomorphic to Pascal's triangle of some height. The figure on the right shows a pyramid DAG of *height* 4. Pyramid DAGs occur in the complete Horner scheme [Dow90], in applications of dynamic programming [Gen96, GP94, HLV92], and elsewhere. The problem of mapping the DAG-nodes to P processors has received attention in the literature. Kumar et al. [KK93] propose a shuffling method which partitions the DAG into cells of P nodes each, see Figure 1(a). Lewandowski et al. [LCB96] compare two other mapping methods for the *diamond DAG*, a graph obtained by identifying the bottom nodes of a pyramid DAG with those of an inverted pyramid DAG. The *pipelining* method partitions the DAG into diagonal stripes, see Figure 1(b). The *diagonal method* partitions the DAG into horizontal stripes and distributes about $1/P$ of the entries in each stripe to each of the processors; at the end of each stripe all processors synchronize. Lewandowski et al. conclude that pipelining outperforms the diagonal method due to imbalances in the synchronization phases. Boeres et al. [BCT95] analyze the applicability of the Earliest Task First (ETF) heuristic [HCAL89] to the diamond DAG. As in pipelining, entire stripes are assigned to the processors. The resulting schedule is optimal if the latency is smaller than the node computing time, and if the overhead of sending a message is ignored.

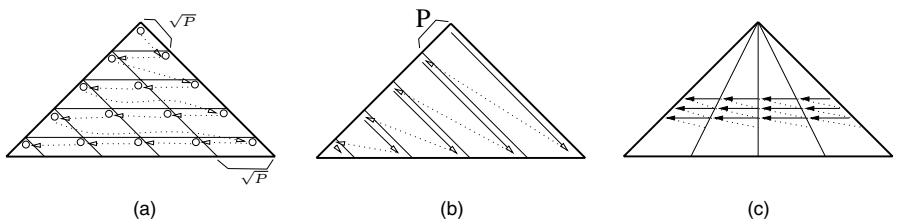
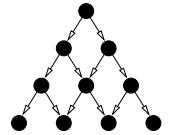


Fig. 1. *Mappings for the pyramid DAG.* (a) *Shuffling:* Each processor computes one node in each of the $\sqrt{P} \times \sqrt{P}$ -cells. (b) *Pipelining:* Processor p , $p \in 0, \dots, P - 1$, computes all diagonals i with $i \bmod P = p$. (c) *Pie mapping:* Each processor computes one sector—level by level and from right to left; communication takes place across sector boundaries.

Since message passing systems such as PVM or MPI produce considerable overhead we take this extra cost into account. We show that the communication overhead of the diagonal method is only linear in the height of the pyramid while shuffling and

pipelining generate quadratic communication overhead. We do this using the *LogP*-model [CKP⁺93]. The four parameters L , o , g , and P describe system properties. Each message takes L time units to traverse the network. The processors are blocked for o cycles while they are sending or receiving messages; no other computations can be done during this time. The parameter g is used to model the bandwidth of the network.

Theorem 1. *Let m be the height of the pyramid DAG and let t_c be the computing time for each DAG-node. Then the computing times of shuffling, $T_P^{(shuffle)}$, and of pipelining, $T_P^{(pipeline)}$, are bounded as follows.*

$$T_P^{(shuffle)} \leq \frac{1}{2} \left(\frac{m^2}{P} + \frac{m}{\sqrt{P}} + P + \sqrt{P} \right) (t_c + 4o) + (m - 1)L \quad (1)$$

$$T_P^{(pipeline)} \leq \frac{1}{2} \left(\frac{m^2}{P} + m \right) t_c + \left(\frac{m^2}{P} + m - \frac{m}{P} - 1 \right) o + (m - 1)L \quad (2)$$

We omit the proof for lack of space. With either method, the communication overhead is of the same order as the computing time. Our variant of the diagonal method is called *pie mapping*. Figure 1(c) sketches the idea; Figure 2 provides details.

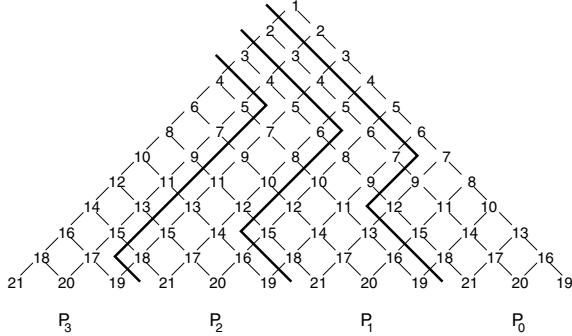


Fig. 2. *Pie mapping for 4 processors and $m = 12$. The figure shows the dependencies among the DAG-nodes and the mapping of the nodes to the processors. Each node is represented by the number of the time-step at which its computation is finished; the numbers show that the schedule respects the data dependencies.*

Theorem 2. *Assume that the nodes of the pyramid DAG are mapped to processors $0, \dots, P - 1$ using pie mapping, assume that each node requires computing time t_c , and assume that the height of the pyramid DAG is $m = sP$ for some $s \in \mathbb{N}$. Then the computing time $T_P^{(pie)}$ is bounded as follows.*

$$T_P^{(pie)} \leq \begin{cases} \frac{1}{2} \left(\frac{m^2}{P} + \frac{m}{P} + P - 1 \right) t_c + (m - 1)(4o + L) & \text{if } s \text{ is odd,} \\ \frac{1}{2} \left(\frac{m^2}{P} + \frac{m}{P} + 2P - 2 \right) t_c + (m - 1)(4o + L) & \text{if } s \text{ is even.} \end{cases} \quad (3)$$

Again, we omit the proof. In our implementation of the pie mapping we allow processors to deviate from the order Figure 2 prescribes for processing the DAG-nodes. Each processor tries to proceed line by line and from right to left, but whenever the next DAG-node is not ready (due to a missing input), then other—ready—DAG-nodes are processed.

All three methods—shuffling, pipelining, and pie mapping—reach efficiencies close to 1 when the node computing time t_c is large. Inequality (3) shows that the pie mapping requires only linear communication overhead; this leads to a better scaling behavior.

Figure 3 shows efficiencies that were obtained using a benchmark program that generates fixed computation and communication costs. The pie mapping outperforms the two other methods in every situation.

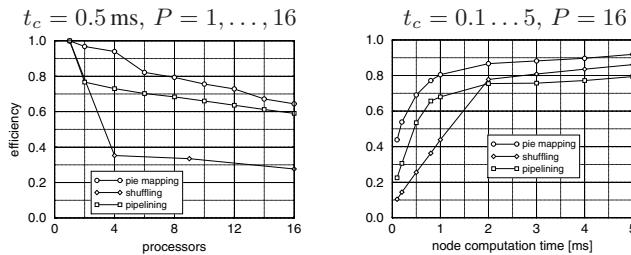


Fig. 3. Efficiency of the various mappings. The left diagram shows that, for small node computing times, pipelining and pie mapping scale better than shuffling. The right diagram shows that pipelining is the least efficient method when node computing times are large. This is due to idle-times at the beginning of each diagonal. In all cases, pie mapping outperforms the two other methods. The experiments were conducted on the Parsytec CC, a massively parallel system using 133-MHz PowerPC 604 processors connected by a fat mesh of clos (2×24).

A detailed analysis of the pie mapping enables us to predict its runtime for arbitrary values of P when m and t_c are given.

3 Scheduling Parallelizable Independent Tasks

In this section we consider a load balancing problem where the processors may outnumber the tasks. We assume that each task is parallelizable and that all tasks have the same computing time function $t(p)$ in the number of processors. We also assume that $t(p)$ is known and that each processor may work on only one task at a time. If N is the number of tasks and P the number of processors, we want to find a non-preemptive schedule that assigns every task a start time and a set of processors.

Instead of computing an optimal schedule we compute a schedule that is optimal among the *phase-parallel schedules*; these are schedules that consist of a sequence of parallel phases. Let m be the number of phases, and let $N = n_1 + \dots + n_m$ such that $n_i \leq P$ for $i \in \{1, \dots, m\}$. Phase i processes n_i tasks on n_i disjoint sets of processors; each set consists of $\lfloor P/n_i \rfloor$ processors. We denote the schedule by (n_1, \dots, n_m) ; its makespan is $\sum_{i=1}^m t(\lfloor P/n_i \rfloor)$.

To find an optimal phase-parallel schedule we exploit the fact that an optimal, non-trivial schedule is a concatenation of two optimal schedules. If $N \geq P$, we take the

Algorithm: Phase-parallel scheduling

Input: $P = \text{number of processors}$, $N = \text{number of tasks}$, $N < P$
Output: a phase-parallel schedule (n_1, n_2, \dots, n_m) with minimum makespan

1. for i from 1 to N
2. $S[i] \leftarrow (t(\lfloor \frac{P}{i} \rfloor), (i))$
3. for j from 1 to $\lfloor \frac{N}{2} \rfloor$
4. if $S[j].t + S[N-j].t < S[i].t$ then
5. $S[i] \leftarrow (S[j].t + S[N-j].t, S[j].S \circ S[N-j].S)$
6. return $S[N].S$

Algorithm 1. The algorithm schedules N jobs on P processors. The dot-notation denotes access to record entries: if $S[i] = (x, y)$, then $S[i].t = x$ and $S[i].S = y$. The \circ operator concatenates two schedules.

$\lfloor N/P \rfloor$ -phase schedule (P, P, \dots, P) and concatenate it with a schedule for $N - P\lfloor N/P \rfloor$ tasks. Algorithm 1 uses dynamic programming to construct a schedule for $N < P$ tasks.

Theorem 3. Let P be the number of processors and $N < P$ the number of tasks. Then Algorithm 1 computes a phase-parallel schedule with minimum makespan in time $O(N^2)$.

As before, we omit the proof. We note that a more general form of the problem is known to be intractable. If each task has an arbitrary computing time, finding an optimal schedule is strongly NP-hard for $P \geq 5$ [DL89]. Krishnamurti et al. [KM92] give an algorithm for $N < P$ that produces a schedule that is optimal among all *parallel schedules*. Those are schedules in which all tasks start executing at time 0. The authors give an upper bound for the ratio T_a/T_o between the makespan T_a of an optimal parallel schedule and the makespan T_o of an optimal schedule; if $r := \max_{1 \leq i \leq N, 1 \leq p < P} t_i(p+1)/t_i(p)$, $T_a/T_o \leq \min\{P, r/(1-N/P)\}$.

4 Parallel Real Root Isolation

The Descartes method performs a binary search; it starts with an interval that is known to contain all real roots; then it proceeds by interval bisection using Descartes' rule of signs to decide whether an interval contains no real root or exactly one real root. The initial interval is obtained by computing a root bound. In the search tree, Taylor shifts are used to construct the right children of internal nodes and to decide whether a node is a leaf. Each internal node requires two or three Taylor shifts by 1. A Taylor shift by 1 computes the coefficients of the polynomial $B(x) = A(x+1)$ from the coefficients of a polynomial $A(x)$. If $A(x+1)$ is computed by classical methods, the intermediate results form a pyramid DAG of height $\deg(A) + 1$ whose nodes represent additions of scalars. For efficiency we coarsen the DAG by covering it with tiles of about the same size and by considering the pyramid DAG formed by the tiles.

We experiment with three kinds of input polynomials. *Random polynomials* are polynomials whose coefficients are 2000-bit integers that are generated uniformly at

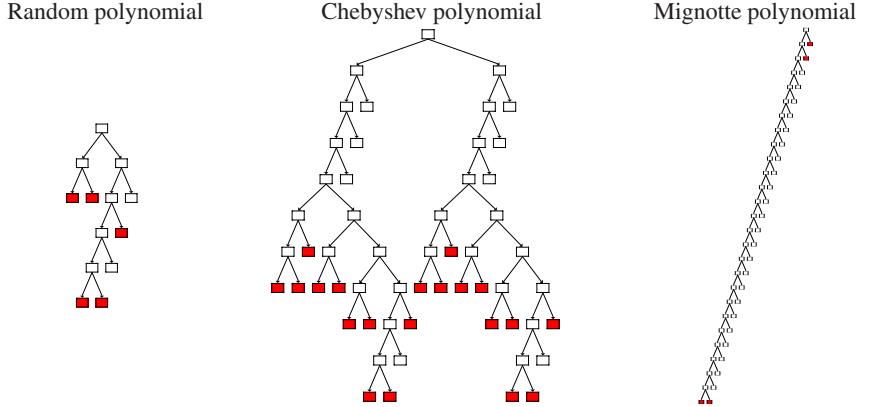


Fig. 4. Search trees arising from isolating the real roots of polynomials of degree 20. Shaded nodes correspond to isolating intervals.

random. Search trees for random polynomials tend to be small. *Chebyshev polynomials* have only real roots. Chebyshev polynomials (of the first kind) are recursively defined as $T_0(x) = 1$, $T_1(x) = x$, $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. Search trees for Chebyshev polynomials are wide and deep. *Mignotte polynomials* are defined as $A_n(x) = x^n - 2(5x - 1)^2$. Search trees for Mignotte polynomials are narrow and extremely deep. Figure 4 shows examples of search trees.

Our implementation is based on the load balancing system *VDS* [Dec97]. We integrated the various scheduling methods for DAGs and the phase-parallel scheduler. We also used the system to generate execution profiles.

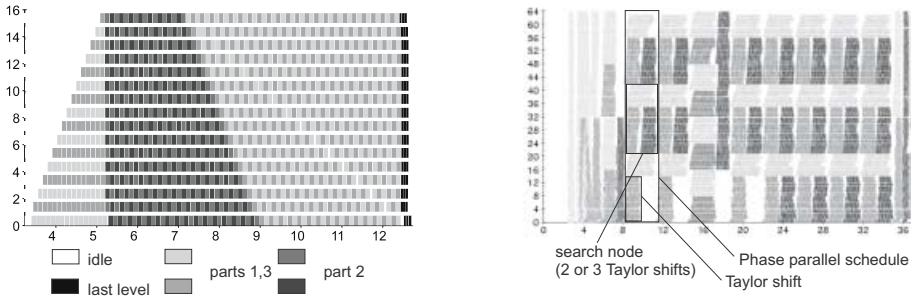


Fig. 5. Execution profiles of a Taylor shift (left) and of a root isolation (right). The horizontal axis is the time axis; the vertically stacked horizontal bars show the activities of the processors. White indicates idle time; grey shades indicate computation. The shades in the left diagram distinguish DAG-nodes; the shades in the right diagram distinguish the Taylor shifts associated with each search-node. The left profile was obtained on a Parsytec CC, the right profile on a workstation cluster of 64 Siemens Primergy server nodes (Pentium II, 450 MHz, PVM).

The left diagram in Figure 5 shows the execution profile of a parallel Taylor shift of a polynomial of degree 8000 executed on $P = 16$ processors. The (coarsened) DAG has

height 48; thus, in terms of Theorem 2, $s = 3$. The left part showing light grey shades corresponds to the processing of levels 1 through P , the darker part in the middle to levels $P + 1$ through $2P$, and the light grey part on the right to levels $2P + 1$ through sP . Note that the processors finish the odd parts at the same time.

The right diagram in Figure 6 shows that our parallel version of the Descartes method scales well for random polynomials and particularly well for Chebyshev polynomials. Mignotte polynomials are difficult because their search trees are very deep and narrow. The depth leads to a large amount of work; the lack of parallelism in the tree means that the execution degenerates to a sequence of (parallel) Taylor shifts.

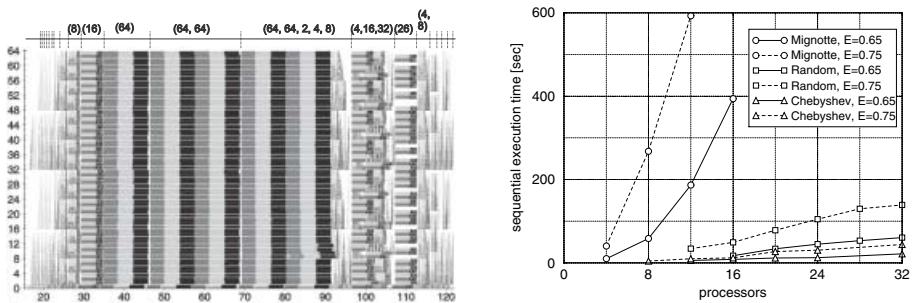


Fig. 6. The left diagram shows a root isolation profile for the Chebyshev polynomial of degree 500; it was obtained on a Cray T3E using 64 processors (MPI). The legend on top of the diagram indicates the levels of the search tree and some phase-parallel schedules. Since Chebyshev polynomials give rise to very wide search trees, most of the Taylor shifts are executed on a single processor or on a small number of processors. This leads to high efficiencies for this class of polynomials. The right diagram shows isoefficiency graphs [GGK93] for the three classes of polynomials. An isoefficiency graph shows the amount of total work necessary to obtain a given efficiency E ; here, $E \in \{0.65, 0.75\}$.

5 Conclusion

Scheduling methods for DAGs and parallelizable independent tasks with identical computing time functions lead to an efficient parallelization of the Descartes method for polynomial real root isolation. When the method is run on 16 processors of the Cray T3E, it takes 1.2 seconds on the average to isolate the real roots of random polynomials of degree 1000. We are now working on a parallel method for refining isolating intervals to any desired accuracy. Future challenges are the isolation of polynomial complex roots and the refinement of isolating rectangles.

References

- [BCT95] C. Boeres, G. Chochia, and P. Thanisch. On the scope of applicability of the ETF algorithm. In A. Ferreira and J. Rolim, editors, *Proc. of the 2nd International Symp. on Parallel Algorithms for Irregularly Structured Problems*, number 980 in Lecture Notes in Computer Science, pages 159–164. Springer, 1995.

- [CA76] George E. Collins and Alkiviadis G. Akritas. Polynomial real root isolation using Descartes' rule of signs. In R. D. Jenks, editor, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 272–275. ACM, 1976.
- [CJK92] G. E. Collins, J. R. Johnson, and W. Küchlin. Parallel real root isolation using the coefficient sign variation method. In R. E. Zippel, editor, *Computer Algebra and Parallelism. Second International Workshop. Ithaca, USA, May 1990*, number 584 in Lecture Notes in Computer Science, pages 71–87. Springer-Verlag, 1992.
- [CKP⁺93] D. Culler, R. Kark, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12. ACM, 1993.
- [Dec97] T. Decker. Virtual Data Space—A universal load balancing scheme. In G. Bilardi, A. Ferreira, R. Lüling, and J. Rolim, editors, *Proceedings of the 4-th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1253 of *Lecture Notes in Computer Science*, pages 159–166. Springer, 1997.
- [DL89] J. Du and Y-T. Leung. Complexity of scheduling parallel tasks systems. *SIAM Journal of Discrete Mathematics*, 2(4):473–487, 1989.
- [Dow90] Michael L. Dowling. A fast parallel Horner algorithm. *SIAM Journal on Computing*, 19(1):133–142, 1990.
- [Gen96] M. Gengler. An introduction to parallel and dynamic programming. In A. Ferreira and P. Paradalos, editors, *Solving Combinatorial Optimization Problems in Parallel*, volume 1054 of *Lecture Notes in Computer Science*, pages 87–114, 1996.
- [GGK93] A. Grama, A. Gupta, and V. Kumar. Isoefficiency function: A scalability metric for parallel algorithms and architectures. *IEEE Parallel and Distributed Technology, Special Issue on Parallel and Distributed Systems: From Theory to Practice*, 1(3):12–21, Aug. 1993.
- [GP94] Z. Galil and K. Park. Parallel algorithms for dynamic programming recurrences with more than $o(1)$ dependency. *Journal of Parallel and Distributed Computing*, 21:213–222, 1994.
- [HCAL89] J-J. Hwang, Y-C. Chow, F.D. Anger, and C-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing*, 18(2):244–257, 1989.
- [HLV92] S.-H. S. Huang, H. Liu, and V. Viswanathan. A sublinear parallel algorithm for some dynamic programming problems. *Theoretical Computer Science*, 106:361–371, 1992.
- [KK93] G. Karypis and V. Kumar. Efficient parallel formulations for some dynamic programming algorithms. In *Proc. of the 7th International Parallel Processing Symposium (IPPS)*, 1993.
- [KM92] R. Krishnamurti and E. Ma. An approximation algorithm for scheduling tasks on varying partition sizes in partitionable multiprocessor systems. *IEEE Transactions on Computers*, 41(12):1572–1579, dec 1992.
- [Kra95] Werner Krandick. Isolierung reeller Nullstellen von Polynomen. In J. Herzberger, editor, *Wissenschaftliches Rechnen*, pages 105–154. Akademie Verlag, Berlin, 1995.
- [LCB96] G. Lewandowski, A. Condon, and E. Bach. Asynchronous analysis of parallel dynamic programming algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 7(4):425–437, 1996.

Cellular Automata Based Transform Coding for Image Compression

Kolin Paul¹, D. Roy Choudhury², and P. Pal Chaudhuri¹

¹ Department of Computer Science & Technology
Bengal Engineering College (Deemed University)
Howrah, West Bengal, India

² Department of Computer Science & Engineering
Indian Institute of Technology
Kharagpur, West Bengal, India

Abstract. In this correspondence, we propose a new Cellular Automata based transform coding scheme for grey level and color still images. This new scheme is markedly superior to the currently available DCT based schemes both in terms of *Compression Ratio* and *Reconstructed Image Fidelity*.

1 Introduction

* This paper sets a new direction in the application of **Cellular Automata** (CA) technology for image compression. A large number of image compression algorithms have been reported in last few years [1]. A new transform, termed as CA based Transform (CAT) coding has been reported in this paper to achieve the above goals for image compression. Compared to the current state of the art algorithms, this scheme is computationally simple and **achieves superior reconstructed image quality at higher compression ratios**.

2 CA Preliminaries

Cellular Automata are mathematical idealizations of physical systems in which space and time are discrete, and physical quantities take on a finite set of discrete values. A cellular automata consists of a regular uniform lattice (or "array") with a discrete variable at each site("cell"). The state of a cellular automata is completely specified by the values of the variable at each site. A cellular automaton evolves in discrete time steps, with the value of the variable at one site being affected by the values of the variables in its "neighborhood" on the previous time step. The cellular automaton evolution can be expressed in the form

$$b_{i,(t+1)} = f(b_{(i-1),t}, b_{i,t}, b_{(i+1),t}) \quad (1)$$

* This work has been partially supported by a research project funded by Silicon Automation Systems, Bangalore, India.

In [2] we have studied 3 neighbourhood $GF(2^p)$ CA where each cell may have a value in the extension field from 0 to $(2^p - 1)$. Encouraging results have been obtained from the application of two dimensional $GF(2^p)$ CA theory in the field of image video processing. However for the sake of simplicity, we shall mainly use one dimensional $GF(2)$ CA framework in subsequent discussions.

3 Transform Coding

The underlying principle of transform coding can be stated as follows. A set of data sample values is taken and the basic objective of the transformation is to alter the distribution of the values representing the luminance levels so that most of the values can either be ignored or can be quantized with a very few number of bits. The basic objective of the transform process is to perform the operation

$$\mathbf{d}_i = \sum_j c_j \mathbf{B}_{ij} \quad (2)$$

where the values \mathbf{d}_i are the values of the image data in the spatial domain while c_j are the values of the data in the transform domain and B_{ij} is the underlying basis function of the transform domain. We perform this transformation because the set c_j is more amenable to further processing to realize the specified objective.

4 CA Based Transform Coding (CAT)

The basic principle of CAT is introduced in this section. We are given a physical process described by a set of discrete values say γ_i . This function is defined in a cellular space (grid) i . We want to express γ_i in the following manner

$$\gamma_i = \sum_j c_j \mathbf{B}_{ij} \quad \forall i \quad (3)$$

where \mathbf{B}_{ij} are the basis functions and c_j are the associated transform coefficients defined in the CA space j . The basis functions are related to the evolving field of the cellular automata.

One dimensional $GF(2)$ cellular spaces offer the simplest environment for generating CA transform with a small number of bases. It is also possible to generate two dimensional $GF(2^p)$ CA bases from combinations of one dimensional base elements. In a one-dimensional space consisting of N cells, the transform base is

$$\mathbf{B}_j \equiv \mathbf{B}_{ij} \text{ for } i, j = 0, 1, \dots, (N-1) \quad (4)$$

For the data sequence $\gamma_i (i=0,1,2,\dots,(N-1))$ we have

$$\gamma_i = \sum_{j=0}^{N-1} c_j B_{ij} \quad (5)$$

where $\{c_j\}$ are the transform coefficients. There are a host of ways in which B_{ij} can be expressed as a function of $a \equiv a_{it}$, (i,t=0,1,2,...,(n-1), a_{it} being the state of the i^{th} cell at t^{th} instant of a n cell CA. The basis function therefore directly depends on the evolving field of the underlying Cellular Automata. We have used the following basis functions as our transform bases.

$$B_{ij} = 2a_{ij} - 1 \quad (6)$$

$$B_{ij} = 2a_{ij}a_{ji} - 1 \quad (7)$$

$$\begin{aligned} B_{ij} &= \rho_{ij}\rho_{ji} \\ \rho_{ij} &= 2a_{ij}a_{ji} - 1 + \rho_{ij-1} \\ \rho_{i0} &= 2a_{ji} - 1 \end{aligned} \quad (8)$$

Two Dimensional Bases

In a 2D square space consisting of $N \times N$ cells, the transform base $\mathbf{B}_j \equiv B_{ikjl}$ i,k=0,1,2.....(N-1). For the data sequence γ_{ik} (i,k=0,1,2....(N-1) we have

$$\gamma_{ik} = \sum_{j=0}^{N-1} \sum_{l=0}^{N-1} c_{jl} B_{ikjl} \quad i,j = 0,1,2,\dots,(N-1) \quad (9)$$

in which c_{jl} are the transform coefficients.

5 Proposed Algorithm

In this section we detail our scheme. The basis function B_{ij} described in the above section is obtained in our case with a very simple rule.

$$B_{ij} = 2a_{ij} - 1 \quad (10)$$

The rule by which a_{ij} is obtained, may be stated simply as

$$a_i^{t+1} = \overline{a_{i-1}^t} \quad (11)$$

where \overline{a} denotes boolean complementation. The rule simply says that the present state of a particular cell in the cellular automaton is the complement of the state of left adjacent cell in the previous time step. The CA may be a one dimensional CA or a two dimensional CA. For purposes of convenience, we explain the basic algorithmic steps with respect to a one-dimensional CA.

The algorithm may be written in pseudo-code as follows.

Algorithm 1 Compression Algorithm

INPUT:Raw Image Data File

OUTPUT:Compressed Data File

1. Partition the image into 16×16 blocks.
2. Apply equation ?? on each of these blocks to obtain the transformed data file.

3. Perform a simple quantization — the rule that we followed was that coefficients less than a user defined threshold are discarded.
4. Perform LZ coding on this quantized coefficients file to obtain the final compressed data file.

Algorithm 2 Decompression Algorithm

INPUT: Compressed Data File

OUTPUT: Raw Image Data File

1. Perform LZ decoding on this compressed data file to obtain the file containing quantized coefficients.
2. Perform the dequantization step to restore the transformed coefficients.
3. Apply equation 5 on each of these transformed coefficients.
4. Rearrange the data blocks so obtained to get the final decoded image.

6 Experimental Results

A scheme for the efficient and fast compression/decompression of the color images has been presented. We have tested our compression/decompression algorithms with a large number of sample images most of which are downloaded from the Internet. The algorithms are coded in C and C++ and run on SUN ULTRA 1. The results of running the algorithm are shown in Table 1 which gives the compression ratio and the error metric — PSNR (in dB) obtained using the relation

$$PSNR = 20 \log_{10} \left[\frac{255}{RMSE} \right] \quad (12)$$

for an image with (0-255) levels. Here $RMSE$ is the Root Mean Square Error and is defined by

$$RMSE = \frac{1}{N} \sqrt{\sum_{i=1}^N \sum_{j=1}^N [f(i, j) - \hat{f}(i, j)]^2} \quad (13)$$

where the original $N \times N$ image is denoted by f and the decompressed image is denoted by \hat{f} .

Table 1: Results

Sl. No	Image	Comp Ratio CAT	PSNR	Compr Ratio DCT	PSNR
1.	Wales	84	37	76	33
2.	New_zealand	81	36	80	38
3.	Challenger	80	39	84	35
4.	Tree	83	42	80	33
5.	Jellybeans	90	32	85	32

At lower compression ratios (75%) JPEG outperforms our algorithm with respect to compression ratio and acceptable PSNR and

image quality. But as we increase the compression ratio, the reconstructed image quality of JPEG falls appreciable while that using CAT remains more or less the same.

7 Conclusion

We have presented a new image scheme for the compression of still images that provides very good compression ratio with good reconstructed image quality.

References

1. R. J. Clarke, *Transform Coding of Images*. Addison Wesley, 1985.
2. K. Paul, *Theory and Application of GF(2^p) Cellular Automata*. PhD thesis, B. E. College , (Deemed University), Howrah, India, 1999.

Appendix

In the figures below, the leftmost figure is the original, the next one is the JPEG compressed file and the last one of each tuple is the one obtained using CAT.



Fig. 1. Ireland



Fig. 2. Lena

A Parallel Branch-and-Bound Algorithm for the Classification Problem

Stefan Balev^{1,2}, Rumen Andonov¹, and Arnaud Freville¹

¹ Université de Valenciennes et du Hainaut-Cambresis LAMIH/ROI UMR 8530
BP 311 - Le Mont HOUY - 59304 - Valenciennes Cedex - FRANCE
`{sbalev,andonov,freville}@univ-valenciennes.fr`

² Sofia University, Faculty of Mathematics and Computer Science
Blvd. J. Bourchier 5, 1164 Sofia, Bulgaria

Abstract. The classification problem involves classifying an observation into one of two groups based on its attributes. Determination of a hyperplane which misclassifies the fewest number of observations from the training sample is a hard combinatorial optimization problem and the sequential algorithms for it are still not practical for large size instances. We propose a parallel branch-and-bound algorithm for the problem on distributed-memory MIMD architectures. It achieves a good load balance and a small communication overhead by using a simple load balancing scheme. Our approach is validated by experimental results.

1 Introduction

The problem of classifying an observation into one of a number of groups is a fundamental problem of the scientific inquiry with many applications. In the two-group linear discriminant analysis, the groups are separated by a hyperplane determined from a training sample – a set of observations, whose group membership is *a priori* known. When the objective is to minimize the number of the misclassified observations, we have a hard combinatorial problem. Even the most efficient sequential algorithms for it [1, 3, 9] are not practical for large instances which motivates the development of parallel algorithms.

The performance of parallel B&B algorithms depends on the efficiency of their work-splitting and load balancing strategies [2, 4]. Many parallel search libraries which implement a variety of these strategies are readily available. Although they simplify the development, they are rather general and maintain complex data structures, which increases the overhead of the parallel algorithms. Instead of using them, we develop a problem-specific parallel implementation which uses very simple data structures and communication scheme.

2 Problem Formulation and Sequential Algorithm

In this section we briefly sketch the sequential depth-first B&B algorithm from [9], since our parallel implementation is based on it. Our training sample consists

of observations $X_i \in R^m$, $i = 1, \dots, n$ from two groups. The objective is to remove minimal number of them, so that the rest become linearly separable. After appropriate transformations the problem can be formulated as follows: Remove a minimal number of variables from the system

$$Ax = b, \quad x \geq 0 \quad (1)$$

so that it becomes infeasible. Here A is a $(m + 1) \times n$ matrix, each column of which corresponds to an observation.

It is clear that each basic representation of (1) contains a variable that must be removed in order to achieve infeasibility. The branching strategy uses this fact. The root of the branching tree is a feasible basis of (1). Each child of a node is obtained by removing a basic variable and restoring the feasibility.

Suppose we have a sequence of mutually exclusive feasible bases of (1). Since each of them contains a variable to be removed, their number is a lower bound of the solution of the problem. Once a sequence of mutually exclusive bases has been found for the root, it can be used for the other nodes with slight modifications. The sequential algorithm uses this mechanism of generating the lower bounds.

The algorithm starts with a heuristic, which finds a good initial solution.

3 Parallel Algorithm

Two features of the sequential algorithm are important for its parallelization. First, this is the *good initial solution*. The experiments show that the local search heuristic from [9] finds the optimal or near-optimal solution. The good initial solution ensures that the number of the nodes in the search tree is to a certain extent independent on the way the tree is traversed. This allows avoiding the work overhead which is typical for the parallel B&B algorithms (see [5, 6, 8]).

The second important property is that *the level of each node is equal to the value of the objective function in it*. (Recall that a node is obtained from its parent by removing a variable and, in the other hand, the objective is to minimize the number of the removed variables.) The pruning criterion for each node is:

$$\text{level} + l \geq \text{record}$$

where l is the lower bound for the node. Since the record changes occasionally, two nodes at the same level differ only by l . But the lower bound of each node is generated using the lower bound of the parent, hence this difference is not great. In other words, the most of the terminal nodes are concentrated in a narrow strip. Therefore the work amounts in the subtrees at the same level will not be very different. This ensures a good load balance for the parallel implementation.

Our parallel algorithm is appropriate for any distributed-memory MIMD architecture allowing all-to-all communications. It is based on the master-slave paradigm. If there are p processors, we run a master process P_0 and slave processes P_1, \dots, P_p . One of the processors runs the master and one of the slaves concurrently. Fig. 1 shows the communication between the processes.

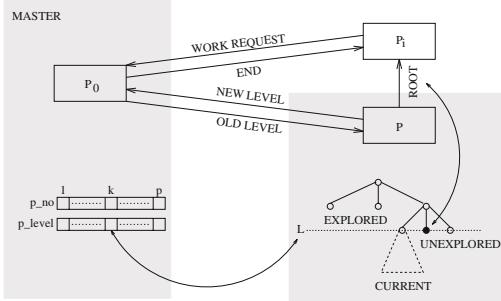


Fig. 1. Parallelization scheme

Each slave explores a subtree of the search tree. The master distributes the work among the slaves. It keeps the information about them using only two arrays of integers – p_level and p_no , where $p_level[k]$ is the level of the highest unexplored node of the subtree of P_k and $p_no[k]$ is the number of the nodes on this level which are already given to some other process.

At the beginning the master assigns the root of the tree to one of the slaves, say P_1 . It sets $p_level[1]=1$, $p_no[1]=0$ and marks the rest as idle, by setting their p_level to some sufficiently large constant. Then it starts waiting for WORK REQUEST messages. Suppose it receives a WORK REQUEST from P_i , which means that P_i is idle. Then the master selects a donor for it. The donor is the process with the highest unexplored node, i.e. P_k with minimum $p_level[k]$. If there are more than one such processes, the master selects the one of them with the smallest $p_no[k]$. The idea is to select the process which has the greatest piece of work available and if there are more than one possibilities, to leave as many work as possible to the donor. If all of the slaves are idle, then the master sends END message to all of them and stops. Otherwise, let $p_level[k]=l$. Since the actual state of P_k may have changed and $p_level[k]$ may not be up-to-date, the master sends a message with a tag OLD LEVEL and data (i, l) to P_k . Then it waits for a NEW LEVEL message with data l' from P_k , where l' is the actual level of the highest unexplored node in the subtree of P_k . If the level of P_k is up-to-date, i.e. if $l = l'$, then the master increments the $p_no[k]$ counter, sets $p_level[i]=l+1$, $p_no[i]=0$ and proceeds with processing the next WORK REQUEST message. Otherwise it updates the state of the k th process by setting $p_level[k]=l'$ and $p_no[k]=0$ and repeats the donor selection procedure.

At the beginning each slave sends a WORK REQUEST to the master and waits for a ROOT or END message. If an END message arrives, it stops, otherwise it extracts a root of a subtree from the body of the ROOT message and starts a depth-first B&B search in this subtree. After each processed node, it checks if there is an OLD LEVEL message waiting. In this case it extracts i and l from its body and returns to the master l' , the level of its highest unexplored node, by NEW LEVEL message. If $l = l'$, the slave takes its highest unexplored node, marks it as explored and sends a ROOT message to P_i , packing in the body of

the message all the data needed to restore this node. When the slave explores its subtree, it sends WORK REQUEST to the master and starts waiting for ROOT or END again. If someone finds a new record, it broadcasts it to the rest.

Our parallelization scheme is a mix of depth-first and breadth-first search – each slave explores its subtree in a depth-first manner, but when some process finishes its subtree, it continues with one of the highest unexplored nodes. To manage the load balancing, the master uses a simple and easy-to-maintain data structure. It is updated only on demand, which simplifies the communication.

4 Computational Experiment

We tested our algorithm on two types of distributed memory parallel platforms: the Intel Paragon XP/S at IRISA¹ and the network of DEC Alpha workstations at LIFL². The program is written in C language. For the first platform we used the NX communication library and for the second one – the PVM library.

For brevity we report computational results only on four data sets, but they are representative for the general behavior of the algorithm. Three of them are real life examples from [7], and the other one is randomly generated. Table 1 gives the number of observations, n , and their dimension, m , for each of the examples. It also shows the number of nodes in the sequential search tree. This is a good measure for the difficulty of each problem. Generally, this number changes when the program is executed on a different number of processors. But these fluctuations are small because of the good initial solution. So the data in the table give an idea for the number of the nodes for all executions.

On Intel Paragon the code was executed on 1 up to 32 processors. For the smallest data set the speedup increases only up to 15 processors. After that, it stays almost the same. For the “serious” examples the speedup is close to linear and the more the nodes of the tree are, the better the speedup is. The efficiency is bad only for the smallest example, while for the second example it is more than 70% and more than 80% for the other two examples. These are quite satisfactory results for a parallel B&B algorithm. Table 1 shows also the communication overhead (the part of the total execution time that take the communications between the processors). Note that almost all of the loss of efficiency is due to the communication overhead. In other words, there is almost no search overhead.

Because of resource limitations, the experiments on the network of DEC Alpha workstations were made only on up to 8 processors. Although the communications between the workstations are much slower than the external inter-processor communications of Intel Paragon, the results are similar.

Our algorithm is designed especially for the classification problem. However, we believe that some of its ideas, as the simple data structures, on-demand update of the data, the “locally depth, globally breadth” search we used, can find applications in other parallel B&B implementations.

¹ IRISA, Campus de Beaulieu, 35042 Rennes, France

² LIFL, Cite scientifique, 59655 Villeneuve d'Ascq, France

Table 1. Computational Results (S – speedup, E –Efficiency(%), O – communication overhead(%))

Data set	1			2			3			4		
$n \times m$	710 × 8			410 × 6			1000 × 3			280 × 6		
nodes	6,000			30,000			70,000			130,000		
source	real life			real life			random			real life		
# proc.	S	E	O	S	E	O	S	E	O	S	E	O
Intel Paragon												
4	3.9	96.8	3.2	4.0	99.4	0.5	4.0	99.9	0.3	4.1	103.5	0.4
8	6.3	78.6	21.4	7.8	96.9	2.9	7.8	98.0	2.3	6.9	86.0	0.6
12	8.2	68.4	31.5	11.3	93.8	6.1	11.5	96.1	4.1	10.3	85.8	1.1
16	9.4	59.0	40.9	14.3	89.5	10.2	14.8	92.6	7.5	13.6	84.9	1.6
20	10	49.8	49.9	16.5	82.7	16.8	17.7	88.7	11.3	16.9	84.6	2.0
24	10.1	42.3	57.4	18.9	78.8	20.7	20.7	86.2	13.7	20.1	83.9	2.6
28	10.3	36.9	62.7	20.9	74.7	24.7	23.4	83.4	16.5	23.4	83.5	3.1
32	10.2	31.9	67.7	22.0	68.8	30.6	26.1	81.5	18.3	26.5	82.9	3.7
Dec Alpha network												
2	2.0	99.4	0.5	2.0	99.1	0.8	2.0	99.8	0.2	1.9	93.5	6.4
4	4.0	99.0	0.9	3.9	97.4	2.6	4.2	104.7	4.7	3.6	90.2	9.7
6	5.6	93.7	6.2	5.9	98.3	1.6	6.3	105.0	5.0	5.1	85.8	14.2
8	6.7	83.7	16.2	7.7	96.1	3.8	8.0	99.6	0.4	6.9	86.4	13.6

References

- W. Banks and P. Abad, An efficient optimal solution algorithm for the classification problem, *Decision sciences* 22 (1991), 1008–1023.
- V.-D. Cung, S. Dowaji, B. Le Cun, T. Mautor, and C. Roucairol, Concurrent data structures and load balancing strategies for parallel branch-and-bound/A* algorithms, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 30 (1997), 141–161.
- G. Koehler and S. Erenguc, Minimizing misclassifications in linear discriminant analysis, *Decision sciences* 21 (1990), 63–85.
- V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to parallel computing: design and analysis of parallel algorithms*, ch. 8, pp. 299–353, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.
- T. H. Lai and S. Sahni, Anomalies in parallel branch and bound algorithms, *Communications of the ACM* (1984), 594–602.
- G.-J. Li and B. W. Wah, Coping with anomalies in parallel branch-and-bound algorithms, *IEEE Transactions on Computers* C-35 (1986).
- P. Murphy and D. Aha, UCI repository of machine learning databases, 1992, Department of Information and Computer Science, University of California, Irvine.
- V. N. Rao and V. Kumar, On the efficiency of parallel backtracking, *IEEE Transactions on Parallel and Distributed Systems* 4 (1993), 427–437.
- N. Yanev and S. Balev, A combinatorial approach to the classification problem, *European Journal of Operational Research* 115 (1999), 339–350.

Parallel Implementation of Tomographic Reconstruction Algorithms on Bus-Based Extended Hypercube

K. Rajan¹ and L.M. Patnaik²

¹ Department of Physics, Indian Institute of Science, Bangalore 560 012, India

² Microprocessor Applications Laboratory, Indian Institute of Science, Bangalore
`rajan@physics.iisc.ernet.in lalit@micro.iisc.ernet.in`

Abstract. The convolution backprojection (CBP) and Fourier inversion method (FIM) are popular tomographic image reconstruction techniques. However, the time required to reconstruct an image has been a major drawbacks associated with these techniques. We have designed a bus-based extended hypercube (BEH) that combines the positive features of both the bus-based systems and the hypercube. We have executed the CBP and FIM algorithms on a hypercube, extended hypercube and on BEH. The reasons for high speedup of the BEH for image reconstruction tasks could be attributed to fast collective communication and collective computation algorithms of BEH.

1 Introduction

The bus-based shared memory multiprocessor systems are easy to implement. The common bus is the most limiting resource in such a system due to bus congestion. The Hypercube (HC) belongs to the other category, namely distributed memory systems. HC has many positive features. Extended Hypercube (EH) [4], proposed by Mohan et. al solves some of the shortcomings of hypercube. The BEH solves the bus congestion by restricting the number of processors sharing a bus to 5-8, and by hierarchically increasing the number of buses.

Computerized Tomography (CT) is an inevitable tool for medical diagnosis. To reconstruct an image of size $N \times N$, the computational complexity of CBP algorithm [2] is $O(N^4)$, whereas that of the FIM algorithm [3] is only $O(N^3 \log N)$.

This paper is organized as follows. In the second section, we give a brief description of the CBP algorithm. The BEH topology is introduced in section III. Section IV discusses an implementation of the BEH and the mapping of the CBP algorithm onto the BEH topology. The FIM algorithm and its parallel implementation on BEH are discussed in section V. The last section concludes the paper with a few comments on the suitability of the BEH topology for image reconstruction algorithms.

2 Convolution Backprojection (CBP) Algorithm

Let $f(x,y)$ be the linear attenuation coefficient at (x,y) in one fixed plane section of an object. The projection $p(s,\theta)$ of an image $f(x,y)$ is given by,

$$p(s, \theta) = \int_{L(s, \theta)} f du = \sum_{L(s, \theta)} f(x, y) \Delta u \quad (1)$$

where $L(s, \theta)$ is the line $s = x \cos \theta + y \sin \theta$ at a distance s from the origin, and u is the distance along L .

For discrete data, the reconstruction algorithm required to approximate $f_B(k\Delta x, l\Delta y)$, where $0 \leq k \leq (K - 1)$, and $0 \leq l \leq (L - 1)$ from $p(m\Delta s, n\Delta \theta)$ where $0 \leq m \leq (N_D - 1)$ and $0 \leq n \leq (N_{angle} - 1)$, is given by

$$f_B(k\Delta x, l\Delta y) = \Delta \theta \sum_{n=0}^{(N_{angle}-1)} \tilde{p}(k\Delta x \cdot \cos(n\Delta \theta) + l\Delta y \cdot \sin(n\Delta \theta), n\Delta \theta) \quad (2)$$

where \tilde{p} : convolved projection data at an angle $n\Delta \theta$, N_{angle} : number of projections, N_D : number of detector elements and $K \times L$: size of the discretized object.

3 Bus-Based Extended Hypercube

BEH is based on HC and retains the positive features of HC. BEH improves the bus capacity by hierarchically increasing the number of buses.

The basic module of a BEH (Fig. 1) consists of a k -cube and an additional node -the network controller (NC). The nodes that form a k -cube in turn share a common bus. The network controller is also connected to the common bus. The common bus consists of address bus, data bus and control signals. Message passing among the nodes within a k -cube is performed by adopting the hypercube message passing algorithms. The nodes communicate with NC through the common bus. The message passing among nodes at different levels of hierarchy is via one or more NCs.

We use a k -cube with an extra link per node, called cubelet (Fig. 2.) for building larger structures. A k -cubelet sharing a common bus with a network controller forms the basic building block. A BEH consisting of one k -cubelet and an NC is referred to as the basic module BEH($k, 1$). BEH($k, 1$) has 2 levels of hierarchy: the k -cubelet at level 1, and an NC at level 0. A BEH($k, 2$) has 2^k cubelets at level 2, one k -cubelet at level 1, and an NC at level 0. A BEH is characterized by two basic parameters (k, h) where k indicates the dimension of the cube and h the number of levels. Within a cubelet, we have the hypercube connection. Across the cubelet, k PEs in each cubelet are used for connecting all other cubelets in a hypercube format. The 2^k NCs at level 1 are connected in a hypercube fashion using link ports.

Each node in BEH(k,h)-net is specified by a binary number $B=b_nb_{n-1}b_{n-2}\dots b_0$ where $n = \log_2 N$, and N is the number of nodes at the leaf level. Of the $(n+1)$ -bit address, lowest k -bits identify the node within a cubelet. The next k bits identify the cubelet. The NC at level 0 has a 1-bit address, and the NCs/PEs at level 1 have $(k+1)$ -bit address. The MSB bit corresponds to the parent NC address and the k -bit LSBs correspond to node address within a cube. The NCs/PEs at level 2 have $(2k+1)$ -bit address.

Using the NC, interprocessor message traffic of a module gets redistributed into two categories, viz., local communication, and global communication. The broadcast and collective computation are executed in $O(h)$ steps.

4 Implementation of the BEH(3,1)

A bus-based extended hypercube system is built around a cube consisting of 2^k ADSP 21062 DSP chips [1], sharing a common bus. The 2^k nodes share a common bus. The internal memory and input/output processor (IOP) registers of the DSP node are called *multiprocessor memory space*. Multiprocessor memory space is mapped into the unified address space of each of the nodes. All nodes access their internal memory using addresses 0x0000 0000- 0x0007 FFFF. The internal memory space of a node with ID i gets configured at 0x0008 0000 + i (0x0008 0000) in the unified address space. A block of the memory in the unified address space is assigned for broadcast.

4.1 Parallel CBP on a BEH System

The NC at level 0 collects the projection data $p(m\Delta s, \theta_n)$ for each of the angles θ_n , $n=0$ to $(N_{angle} - 1)$, from the measurement system. The N_{angle} projection data are cyclically distributed to P leaf level nodes. The NC at level 1 transfers the projection data to the nodes at the leaf level. The convolution of the projection data is done in the frequency domain. While a node is computing the convolved projection data, NC can download a new set of projection data to the PE.

Each PE reconstructs N/P image rows. The convolved projection data is backprojected on to the image domain to reconstruct the partial result. Each PE contains N/P rows of the reconstructed image from N_{angle}/P number of projection rows. In order to complete the reconstruction of the N/P rows on a PE, we use hypercube communication to exchange the projection data between the PEs. In the first phase, we use $PE_i \leftrightarrow PE_{i+1}$ communication link to exchange the projection data between two neighboring PEs, and using the new set of $N_{projection}/P$ rows to update the N/P image. In the second phase, we use $PE_i \leftrightarrow PE_{i+2}$ communication links. In the third phase, we use $PE_i \leftrightarrow PE_{i+4}$ communication links. N/P image rows in each of the PEs get updated by all the projection data. Finally, the NC collects N/P reconstructed images from each of the P nodes.

The execution times of CBP algorithm for 256×256 and 512×512 images on a BEH system with 1, 8, and 64 processors have been tabulated in Tables 1. Table 1. also tabulates the execution time of the CBP algorithm on an Extended hypercube (EH) and on a hypercube(HC) with 8 and 64 nodes.

5 Fourier Inversion Method (FIM)

The Fourier transform of the projection data at an angle θ yields a central cross section of the Fourier transform of the object function $f(x,y)$. Once we compute the FT of the projection data for all projection angles $\theta_n, n = 0..N_{angle} - 1$, the object function $f(x,y)$ can be computed as the 2-D Inverse FFT of the FT of the projection data. The Fourier data are available on a polar lattice. A polar to cartesian coordinate interpolation is required for FFT computation.

Stark et.al [5] have provided the following relation to interpolate the Fourier samples from polar locations $(\mu\Delta R, n.\Delta\theta)$ to cartesian co-ordinates $(u\Delta X, v\Delta Y)$.

$$F(R, \theta) = \sum_{n=-N/2}^{+N/2} \sum_{k=0}^{N_{angle}-1} F(n/2A, 2\pi k/N_{angle}) \text{sinc}[A(R/\pi - n/A)] \sigma(\theta - 2\pi k/N_{angle})$$

where $\sigma(\theta) = \frac{\sin(N_{angle}/2)\theta}{N_{angle}\sin(\theta/2)}$, $R = \sqrt{u^2 + v^2}$, $\theta = \cos^{-1}(u/R)$, and A is the radius of the circle of support of the object $f(x,y)$.

5.1 Parallel FIM Algorithm on BEH System

The NC at level 0 distributes the projection data to all PEs at the leaf level. Neighboring nodes get a set of N_{angle}/P rows of projection data, where P is the total number of PEs at leaf level. Each PE computes 1-D FFT of the projection data. Using the interpolation relation given in equations 3, each PE also computes the partial cartesian samples. Assume that $P/2$ PEs are assigned the data corresponding to quadrants 1 and 3. Another set of $P/2$ PEs are assigned the cartesian samples of quadrant 2 and 4. Using hypercube communication steps, data are exchanged between the processors, such that $PE_i - PE_{i+3}$ contain upper half of the resultant 2-D FFT and $PE_{i+4} - PE_{i+7}$ contain the lower half of the 2-D FFT. For an $N \times N_{image}$, a 2-D IFFT involves 1-D IFFT of N rows followed by 1-D IFFT of N columns. Each PE carries out 1-D IFFT of N_{angle}/P rows. The results are communicated to the NC. When the NC accumulates the complete results of 1-D IFFT, it distributes the column vectors of the resultant matrix and finally integrates the results.

Tables 1. gives the execution times for Fourier Inversion algorithm on a BEH, EH and on HC with 1, 8 PEs, and 64 PEs.

6 Conclusions

The BEH system supports large number of processors, while retaining the simplicity of a bus-based system. The new system solves the bus congestion problem inherent in bus-based systems. The BEH system is found to execute the image

reconstruction algorithms based on CBP and FIM algorithms efficiently. The better performance of the BEH system can be attributed to efficient collective communication and collective computation.

References

1. Analog Device's Users Manual, ADSP 2106x SHARC (1993).
2. Herman ,G.T. *Image Reconstruction from Projections.*, New York, Academic, 1980.
3. Robert M Lewitt, "Reconstruction Algorithms: Transform Methods," *Proceedings of the IEEE.*, Vol. 71, No. 3, pp. 390-408, March 1983.
4. J. Mohan Kumar, and L. M. Patnaik, " Extended Hypercube: a Hierarchical Interconnection Network of Hypercubes", *IEEE Trans. Parallel Distributed Systems.*, Vol. 3, No. 1, pp. 45-57, Jan 1992.
5. H. Stark, J. W. Woods, and I. Paul and R. Hingorani, " An Investigation of Computerized Tomography by Direct Fourier Inversion and Optimum Interpolation," *IEEE Trans. on Biomedical Engineering.*, Vol. 28, No. 7, pp. 496-505, July 1981.

Table 1

Execution time (in milli seconds) based on CBP and FIM Algorithms

Alg	Image	# PEs	BEH			EH			HC		
			1	8	64	8	64	8	64	8	64
CBP	256x256	Exec time	2600	362	50.1	406	56.5	434	62		
		Speedup	1	7.18	51.9	6.4	46	6	41.9		
CBP	512x512	Exec time	10600	1472	201	1683	232	1776	253		
		Speedup	1	7.2	52.7	6.3	45.7	5.9	41.9		
FIM	256x256	Exec time	1540	217	31	248	35	266	38.5		
		Speedup	1	7.09	49.6	6.2	44	5.8	40		
FIM	512x512	Exec time	6210	887	124	986	138	1052	156.6		
		Speedup	1	7	50	6.3	45	5.9	39.8		

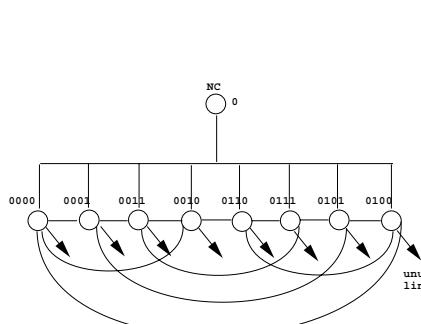


Fig. 1. Basic Building Block of BEH($k,1$)

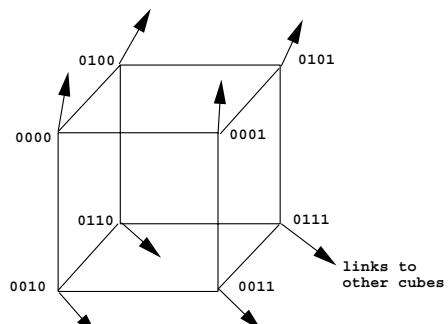


Fig. 2. Cubelet

An Optimal Hardware-Algorithm for Selection Using a Fixed-Size Parallel Classifier Device

S. Olariu¹, M.C. Pinotti², and S.Q. Zheng³

¹ Dept. of Computer Science, Old Dominion University, Norfolk, VA 23529-0162,
USA, olariu@cs.odu.edu

² Istituto di Elaborazione della Informazione, CNR, 56126 Pisa, Italy,
pinotti@iei.pi.cnr.it

³ Dept. of Computer Science, University of Texas at Dallas, Richardson, TX
75083-0688, USA, sizheng@utdallas.edu

Abstract. We present a hardware-algorithm for selecting the k -th smallest item among N elements (for all ranges of N) using a p -classifier device, while strictly enforcing conflict-free memory accesses. Specifically, we show that, by using our design, selection can be accomplished optimally in $O(N/p)$ time.

1 Introduction

Recent advances in VLSI have made it possible to implement algorithm-structured chips as building blocks for high-performance computing systems. With this motivation in mind, we address in this paper the problem of selecting the k -th smallest item among N elements using a classifier device of I/O size p , where N is arbitrary and p is fixed.

In outline, the remaining part of this section discusses the details of our architecture. Section 2 sketches some basic algorithms, but, due to the space constraints, only few of them are devised in details. (The interested reader can find all the details in [3]). The time-optimal algorithm for selection is discussed in Section 3. Finally, concluding remarks and open problems are offered in Section 4.

From now on, we assume that $N \geq p^2$. The basic architectural assumptions (see Fig. 1) of our selection model include:

- (i) A *data memory* organized into p independent, constant-port, memory modules M_1, M_2, \dots, M_p , with p even. Each word is assumed to have a length of w bits, with $w \geq 2 \log p$.

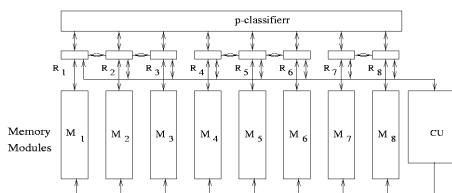


Fig. 1. The proposed architecture for $p = 8$.

- (ii) A set of *data registers*, R_i , ($1 \leq i \leq p$), each capable of storing a $(w + 1.5 \log p)$ -bit word. We refer to the word stored in register R_i as a *composed word*, since it consists of the following three fields (from left to right): (1) an *element field* of w bits for storing an element, (2) a *long auxiliary field* of $\log p$ bits, and (3) a *short auxiliary field* of $0.5 \log p$ bits. Each field of register R_i can be loaded independently from memory module M_i , from the i -th output of the classifier device, or by a broadcast from the CU. The output of register R_i is connected to the i -th input/output of the classifier device, to the CU, and to memory module M_i .
- (iii) A classifier device of fixed I/O size p , with p even, that classifies a set of p values into two classes with the same number of values in such way that each value in one class is at least as large as all of those in the other class. We assume that the outputs of the class of $\frac{p}{2}$ smallest values are connected to the registers $R_1, R_2, \dots, R_{\frac{p}{2}}$.
- (iv) A *control unit* (CU, for short), consisting of a *control processor* capable of performing simple arithmetic and logic operations and of a *control memory* used to store the control program as well as the control data. The CU generates control signals, can broadcast an address or an element to all memory modules and/or to the data registers, and can read an element from any data register.

2 Basic Algorithms

Median. Given a set A of p elements, each one stored into a different memory module, the p -median problem returns the $\frac{p}{2}$ -th smallest element in A .

Theorem 1. *The procedure p -MEDIAN is performed in two calls to the classifier device of I/O size p , and thus, it requires constant time.*

Procedure p -MEDIAN(R , **specified-fields; B , **index**);**

- invoke the classifier device on the **specified-fields** of the data registers;
- for all i , ($1 \leq i \leq p$), set the auxiliary field of R_i to : '10', if $1 \leq i \leq \frac{p}{2}$; '01', if $i = \frac{p}{2} + 1$; '11', if $\frac{p}{2} + 2 \leq i \leq p$.
- invoke the classifier device on the words composed by the short auxiliary and **specified-fields** of the data registers;
- *compare-and-set* the short auxiliary field of $R_{\frac{p}{2}+1}, \dots, R_p$ with the value '10';
- the register R_i storing '10' on its short auxiliary field returns to the CU both its **index** and the value μ of its element field;
- μ is broadcast to all the data registers R and, then, written in the memory row B .

Note that the procedure p -MEDIAN can classify the elements of any combination of the fields of the data registers, specified by the parameter **specified-fields**.

Sorting. The procedure p -SORT invokes p times the procedure p -MEDIAN to build the non-decreasing sorted output sequence in a symmetric way starting from the elements of rank $\frac{p}{2}$ and $\frac{p}{2} + 1$, followed by those of rank $\frac{p}{2} - 1$ and $\frac{p}{2} + 2$, and so on. Before applying the p -median, the last returned item is properly modified so that the new median is the element of the appropriate rank. (See [3] for details.)

Theorem 2. *Procedure p -SORT sorts a memory row of size p in p calls to the procedure p -MEDIAN. Hence, p -SORT takes $O(p)$ time and requires $O(p)$ extra memory rows.*

Partition. Given a splitter μ , and a set A of $\frac{N}{p}$ consecutive memory rows $A_1, A_2, \dots, A_{\frac{N}{p}}$, the procedure PARTITION separates the elements smaller or equal to μ from the ones larger than μ (see [3]).

Theorem 3. *The procedure PARTITION separates with respect to a given splitter μ a set A of $\frac{N}{p}$ memory rows in $T(\frac{N}{p}) = O(\frac{N}{p}) + T(\frac{N}{2p}) = O(\frac{N}{p})$ time.*

Merging. Recently in [1], we have studied hardware-algorithms for sorting N -elements, with $N \geq p^2$, on a special VLSI architecture that uses a sorting device of fixed I/O size p , capable of sorting p elements in constant time. Our architecture, in this paper, is the same as the one in [1] except that the sorting device of size p has been replaced with a classifier device of size p . Since the sorting device can be simulated with the classifier device by the procedure p -SORT, in $O(p)$ time and $O(p)$ space, the hardware-algorithms for sorting studied in [1] can be implemented on our architecture with a slowdown of $O(p)$ both in time and in space. Therefore, the following results hold in our architecture:

Corollary 1. Basic Sorting Algorithm

A set of mp elements stored in m ($1 \leq m \leq p^{\frac{1}{2}}$), memory rows can be sorted, in row-major order, without memory-access conflicts in at most $7m$ calls to the procedure p -SORT and in $O(mp)$ time for data movement not involving sorting. Hence, the task of sorting mp elements requires $O(mp)$ time in our architecture.

Corollary 2. Merge_Two_Groups

The task of sorting $2mp$, ($1 \leq m \leq p^{\frac{1}{2}}$), elements stored in $2m$ memory rows can be performed in five calls to the basic sorting algorithm and $O(mp)$ time.

Corollary 3. Multiway_Merge

The task of merging m , ($2 \leq m \leq p^{\frac{1}{2}}$), sorted sequences, each of size $p^{\frac{i}{2}}$, can be performed using $O(mp^{\frac{i-1}{2}})$ calls to the p -SORT procedure, and $O(m^{\frac{i-1}{2}})$ time for data movement not involving sorting. Overall, the procedure Multiway_Merge requires $O(mp^{\frac{i}{2}})$ time in our architecture.

Ranking.

Theorem 4. [3] *The task of partitioning the input of size $O(\frac{N}{p})$ into $p^{\frac{1}{2}}$ buckets, $B_1, \dots, B_{p^{\frac{1}{2}}}$, each of size at most $2\frac{N}{p^{\frac{3}{2}}}$, can be performed in $O(N/p)$ time and $O(N)$ data memory space by the RANKING procedure.*

Procedure $RANKING(\Sigma_1; B_1, B_2, \dots, B_{p^{\frac{1}{2}}})$;

Step 0. Apply the procedure p -SORT to each memory row of Σ_0 ;

Step 1. – Extract a sample Σ_1 of size $\frac{N}{\frac{1+2}{2}}$ by retaining every $p^{\frac{1}{2}}$ -th element in each sorted sequence of Σ_0 .

- Divide the sorted sequences of Σ_1 in groups, each containing $p^{\frac{1}{2}}$ sorted memory rows. Apply the procedure p -SORT to obtain sorted sequences of size p .
- Divide, again, Σ_1 into groups, each containing $p^{\frac{1}{2}}$ sorted memory rows, and apply the **basic sorting algorithm** to obtain sorted sequences of size $p^{\frac{3}{2}}$.

Step 2. **for** $i := 2$ **to** $t = \frac{\log N}{\log p} - 2$ **do**

- Extract a sample Σ_i of size $\frac{N}{p^{\frac{i+2}{2}}}$ retaining every $p^{\frac{1}{2}}$ -th element in each sorted sequence of Σ_{i-1} .
- The extracted elements are in the form of $\frac{N}{p^{i+1}}$ sorted sequences, each of size $p^{\frac{i}{2}}$, stored in $p^{\frac{i-2}{2}}$ consecutive memory rows. Apply then the procedure **Multiway_Merge** to groups of $p^{\frac{1}{2}}$ sorted sequences, obtaining longer sorted sequences of size $p^{\frac{i+1}{2}}$.
- Apply, again, the procedure **Multiway_Merge** to groups of $p^{\frac{1}{2}}$ sorted sequences, obtaining $\frac{N}{p^{i+2}}$ longer sorted sequences of size $p^{\frac{i+2}{2}}$, stored in $p^{\frac{i}{2}}$ consecutive memory rows.

endfor

Step 3. Terminated the sampling process, the population has shranked into a unique sorted sequence of size $N^{\frac{1}{2}}$. Retain a sample $Q = < q_1, q_2, \dots, q_{\sqrt{p}} >$ with the property that for every i , $(1 \leq i \leq \sqrt{p})$,

$$q_i \text{ is the item of rank } i \left(\frac{N}{p} \right)^{\frac{1}{2}} \text{ in } \Sigma_t. \quad (1)$$

Step 4. **for** $i := 1$ **to** $p^{\frac{1}{2}}$ **to** $PARTITION(\Sigma_0, q_i, B_i, \Sigma_0)$ **endfor**

3 The Selection Algorithm

Procedure $SELECTION(\Sigma, k; x);$

{Input: a set Σ of size N stored in $\frac{N}{p}$ consecutive memory rows, and the rank k of the element sought;

Output: the k -th smallest item x in Σ ; }

Step 1. If $N \leq p^{\frac{3}{2}}$, sort Σ using the **basic sorting algorithm** and returns the element of rank k ;

If $p^{\frac{3}{2}} < N \leq p^2$, first sort individually the $m = \frac{N}{p^{\frac{3}{2}}} \leq p^{\frac{1}{2}}$ sequences of size $p^{\frac{3}{2}}$ in Σ using the **basic sorting algorithm**. Apply the procedure **Multiway_Merge** to the m sorted sequences, and returns the element of rank k ;

Step 2. If $N > p^2$, select the sample Σ_0 of size $\frac{N}{p}$ by retaining from each memory row of Σ its median.

Step 3. Apply the procedure **RANKING** to partition Σ_0 in the buckets

$$B_1, B_2, \dots, B_{p^{\frac{1}{2}}}.$$

Step 4. Counted the elements in B_i , ($0 \leq i \leq p^{\frac{1}{2}}$), we know to which bucket $B_m = \{x \in \Sigma_0 | q_m < x \leq q_{m+1}\}$ the median of Σ_0 belongs. Choose q_{m+1} to act as a splitter in Σ . Apply the *PARTITION* procedure to split the initial set Σ with respect to q_{m+1} into the sets Σ' and Σ'' . Thereby, the exact rank $r^*(q_{m+1})$ of q_{m+1} in Σ is known.

Step 5. If $k = r^*(q_{m+1})$, $x = q_{m+1}$;

If $k > r^*(q_{m+1})$, then *SELECTION* (Σ'' , $k - r^*(q_{m+1})$, x);

If $k < r^*(q_{m+1})$, then *SELECTION* (Σ' , k , x)

Since the median μ belongs to B_{m+1} , and the size of B_{m+1} is at most $\frac{2N}{p\sqrt{p}}$, it holds, as proved in [3]:

Lemma 1. *The rank $r(q_{m+1})$ of q_{m+1} in Σ_1 satisfies, $\frac{N}{2p} \leq r(q_{m+1}) < \frac{N}{2p} + \frac{2N}{p\sqrt{p}}$.*

Moreover,

Lemma 2. *The rank $r^*(q_{m+1})$ of q_{m+1} in the original population Σ yields:*

$$\frac{N}{4} \leq r^*(q_{m+1}) \leq \frac{3N}{4} + \frac{N}{\sqrt{p}} \quad (2)$$

In conclusion, observing that the largest subset of Σ on which Selection may recur has size $O\left(\frac{3N}{4} + \frac{N}{\sqrt{p}}\right)$ and that no more than p elements can be examined in constant time, we have:

Theorem 5. *The worst case overall complexity for selection in Σ is $T(N) = O\left(\frac{N}{p}\right) + T\left(\frac{3N}{4} + \frac{N}{\sqrt{p}}\right)$, for which holds $T(N) = O(N/p)$ for $p > 16$. The Selection algorithm is time-optimal.*

4 Conclusions and Open Problems

This work has been motivated by the observation that scheduling the calls to a p -classifier in such way that selection can be performed on a large set of N elements, with $N > p$, is not trivial. The solution proposed, based on a constant time p -classifier, is time-optimal and accesses the memory without conflicts.

Nonetheless, a lot of work remains to be done. For example, an interesting open question is the existence of a time-optimal selection algorithm, preserving the regular memory access, which does not use the multiway-merge algorithm in [1].

References

1. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison Wesley, Reading, MA, 1973.
2. S. Olariu, M.C. Pinotti and S.Q. Zheng, "An Optimal Hardware-Algorithm for Sorting Using a Fixed-Size Parallel Sorting Device". *10th IASTED Int'l Conference Parallel And Distributed Computing and Systems*, 1998, pp. 38–44.
3. S. Olariu, M.C. Pinotti and S.Q. Zheng, "An Optimal Hardware-Algorithm for Selection Using a Fixed-Size Parallel Classifier Device". *Tech. Rep. IEI-CNR*, 1999.

Session IV-B

Mobile Computing - II

Chair: Ajit Pal

Indian Institute of Technology, Kharagpur

A Novel Frame Structure and Call Admission Control for Efficient Resource Management in Next Generation Wireless Networks *

Naveen K. Kakani¹, Sajal K. Das², and Sanjoy K. Sen³

¹ Department of Computer Science, University of North Texas, Denton, TX 76203.
naveen@cs.unt.edu

² Center for Research in Wireless Computing (CReW), Department of Computer Science & Engineering, University of Texas at Arlington, TX 76019. das@cse.uta.edu

³ Wireless Access Architectures, Nortel Networks, 2201 Lakeside Boulevard, Richardson, TX 75082. sanjoy@nortelnetworks.com

Abstract. We present a dynamic channel assignment algorithm for wireless data networks which satisfies the varied data rate of user services as well as their QoS requirements, while minimally sacrificing the system efficiency. The performance of our scheme is evaluated by computing system *throughput* and *capacity* (i.e., number of users that can be served). We compare our scheme with the IS-136 protocol by simulation experiments. Results show an 800% improvement in the capacity of each slot but at the cost of degradation in system throughput by no more than 67%.

1 Introduction

The field of mobile wireless communications and computing is growing at an ever faster rate. Most of the existing systems are fine tuned to work optimally for voice applications. Given the quality of service (QoS) requirements of next generation applications, it is desirable to have a system that can adapt to user requirements for delay jitter, data rate or bit error rate. Compared to the good old AMPS protocol, the channel utilization of IS-136 protocol for TDMA systems is better since the increase in *capacity* is three times considering the full-rate channel. A channel in IS-136 is divided into 40 msec frames, each in turn subdivided into 6 slots. A full-rate channel uses every third slot in the frame while a half-rate channel uses every sixth slot. Thus, a variable amount of bandwidth can be allocated to users such that one user is allowed to transmit in 1-2 slots per frame. Although the full-rate and half-rate channels support different data rate traffic on the same channel in IS-136, this provisioning scheme does not utilize the full capability of the system resource, namely bandwidth. If the system could adapt based on traffic conditions, the performance would dramatically increase [1,2]. This motivates our work.

In the IS-136 protocol, a user transmitting in one slot in a frame uses the same slot in the next frame. This strategy ensures a definite inter-packet delay

* This work is supported by Texas Advanced Research Program under Award Number TARP-97-003594-013 and also by a grant from NORTEL, Richardson, Texas.

for a certain class of users. However, given the various classes of services that can be requested, some users may not be concerned about the inter-packet delay as long as they are guaranteed an acceptable average data rate of transmission. We take advantage of this concept to reduce the time duration for which the system bandwidth is not in use, thereby allocating one time slot to multiple users.

In our earlier work, we proposed dynamic assignment of service requests to frames, in which the frame structure remains the same [3,4]. In this paper, we propose an algorithm to optimize the system performance under a wide range of QoS requirements of user requests and also allowing the frame structure to change dynamically based upon the airlink frame loss probability. By an interleaving usage of the available bandwidth by different users, we obtain a better utilization of system resources. We evaluate the throughput and the number of users served in each slot in each frame by the system, and also compare our scheme with the IS-136 protocol. The rest of the paper is organized as follows. Section 2 analyzes types of service requests generated by users. Section 3 presents a new slot assignment algorithm and a linear programming formulation for allocating users to a slot. Section 4 presents results of our simulation experiments.

2 Classification of Traffic Requests

Based upon inter-packet delays (τ) and data rates (\mathcal{R}_{user}) of services, we consider two broad classes of traffic, denoted as Mode-1 and Mode-2, as characterized in Table 1. The details of these traffic classes are described in [4].

Table 1. Traffic classes

Class	Mode-1	Mode-2
Inter-packet delay (τ)	Cannot support less than one frame delay	No hard limit. Depends upon the application
User data rates (\mathcal{R}_{user})	Minimum data rate as demanded by the application	Minimum data rate as demanded by the application
Applications	Voice, real time data	Email, fax

Figure 1 illustrates how users are allowed to transmit in various time slots within a frame. The notations used:

- N = total number of slots in a frame
- N_1 = number of slots allocated to Mode-1 users
- N_2 = number of slots allocated to Mode-2 users
- M = number of minislots in each Mode-1 slot
- r = number of retransmission minislots in each Mode-1 slot
- k = number of minislots assigned for data transmission in each Mode-1 slot
- k' = maximum number of interleaved users assigned to each Mode-2 slot
- R = Expected queue length of each minislot user
- D = The bandwidth or the total number of bits that can be sent in one time slot duration
- h = The header length for each of the minislots
- $\hat{h} = \frac{h}{D}$.

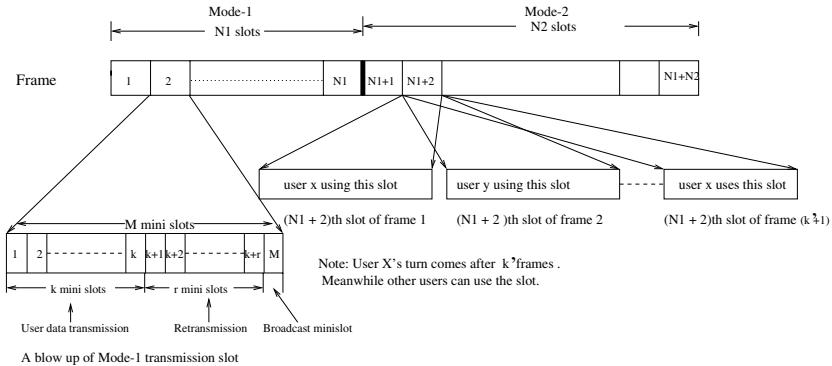


Fig. 1. A snap shot of the frame according to our new algorithm

- All users having Mode-1 traffic transmit in each frame. Each user can transmit in a fraction of a time slot based upon its data rate requirements.
- In Mode-2, only one of the k' users assigned to the time slot, transmits in that slot and subsequently remains idle until its turn comes back after k' frames where k' is determined by the data rate requirements.

Furthermore, we do not allocate a time slot simultaneously to both Mode-1 and Mode-2 traffic.

2.1 Analysis of System Performance

Mode-1 requests: Assume that a user who is allocated multiple minislots, transmits the header information in each minislot. Since k minislots are allocated for user data transmission in each slot, the total number of retransmission slots is given by $r = kR$.

Number of Retransmissions: A user successful in transmitting data in a minislot, will proceed by transmitting a new packet in the same minislot in the next frame. Otherwise, the lost packet is appended to the retransmission list and the next packet is sent in the user minislot. Since the retransmission is accomplished with the help of retransmission minislots, the user has the current minislot available to transmit the next packet.

Figure 2 depicts the state transition diagram of the retransmission list for each slot, where p_l denotes the airlink frame loss probability and $q_l = 1 - p_l$. Typically, the value of p_l depends upon the size of the data being transmitted and the forward error coding (FEC) technique used. A state S_i means the retransmission list has i packets. For example, the list is in state S_0 when there are zero packets to be retransmitted. Since each slot is associated with k minislots for user data, if there is any loss of frame when the system is in state S_0 , the next state of the retransmission list will be state $S_{l \times r}$ under the assumption that the ratio $\frac{k}{r} = l$ is an integer. Assuming that user sends a new packet in each frame, the probability of a transition from state S_0 to state $S_{l \times r}$ is nothing but p_l . If the retransmission list is in state $S_{l \times r}$, the lost packet are sent through the retransmission minislots. At the same time a new packet is sent through the minislot allocated to that user. So the probability for a transition from state

$S_{l \times r}$ to state $S_{2 \times l \times r}$, denoted as $S_{l \times r} \rightarrow S_{2 \times l \times r}$, is given by p_l . Similarly, q_l is the probability for making the transition $S_r \rightarrow S_0$.

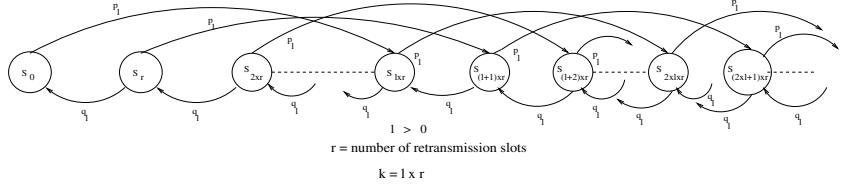


Fig. 2. Modeling retransmission queue (list) of each slot carrying Mode-1 traffic

The steady state probability, P_i , of the retransmission list is derived as:

$$P_i = P_{i-k} p_l + P_{i+r} q_l \text{ for } i \geq k \text{ and } i = a r \text{ where } a \geq l,$$

$$P_i = P_{i+r} q_l \quad \text{for } 0 < i < k \text{ and } i = a r \text{ where } 0 < a < l, \text{ and}$$

$$P_0 = \left(\frac{q_l}{p_l} \right) P_r$$

We analyze the system performance by restricting queue length as $100 \times k$ and solving above equations. In other words, the analysis is based on the assumption that the probability of a user having more than 100 packets accumulated for retransmission is negligible.

Let \mathcal{R}_{min} be the minimum data rate traffic and \mathcal{R}_{header} be the number of header bits expressed as data rate. Now if the slot has a data rate of \mathcal{R}_{slot} , the total number of minislots is given by

$$M = \frac{\mathcal{R}_{slot}}{\mathcal{R}_{min} + \mathcal{R}_{header}}. \quad (1)$$

Now k out of these M minislots are assigned to users, $r = k \times R$ of them are assigned for retransmission (where R is the retransmission queue length for each minislot user), and one of them is assigned as a broadcast minislot. Hence, $M = k + kR + 1$.

The value of R is approximated using a curve fitting software (Mathematica) as $R = 4.69445\left(\frac{r}{k}\right)^2 - (0.5543 + 65.944 p_l)\left(\frac{r}{k}\right) + 0.0417 + 6.37p_l + 190.739(p_l)^2$. Since $M = k + kR + 1 = k + r + 1$, substituting for R and r in terms of M and k leads to

$$M = K_1 \times \frac{(M - k - 1)^2}{k} + K_2 \times k + K_3 \quad (2)$$

where $K_1 = 4.69445$, $K_2 = 1.6 + 72.314125 p_l + 190.74p_l^2$

and $K_3 = -(M - 1)(0.5543 + 65.944125 p_l) + 1$.

Equation (2) is quadratic in k having a solution

$$k = \frac{-\mathcal{B} + \sqrt{\mathcal{B}^2 - 4 \times \mathcal{A} \times \mathcal{C}}}{2 \times \mathcal{A}} \quad (3)$$

where $\mathcal{A} = K_1 + K_2$, $\mathcal{B} = 2K_1 - 2MK_1 + K_3 - M$ and $\mathcal{C} = M^2K_1 - 2MK_1 + K_1$. If \mathcal{R}_{user} is the data rate of the application requested by a user, then the number of allocated minislots is $\lceil \frac{\mathcal{R}_{user}}{\mathcal{R}_{min}} \rceil$. If one of the users of a minislot within a slot terminates its call and if it happens to be the last user for that slot, then the slot is reallocated. Otherwise we let the minislot go idle. Evaluating the steady state probabilities as in [4], the expected number of active slots is given by

$$k_{avg} = \sum_{i=1}^k i \times P_i^{Mode1} \quad (4)$$

where P_i^{Mode1} denotes the probability that i users are assigned to one Mode-1 slot.

Expected Number of Users in Mode-2 Slot: For Mode-2 requests, each user is allocated a time slot in a frame after a certain (fixed) number of frames, denoted by $q = \frac{\mathcal{R}_{slot} - \mathcal{R}_{header}}{\mathcal{R}_{user}}$. The maximum possible number (k') of interleaved frames before a user can retransmit, is given by

$$k' = \frac{\mathcal{R}_{slot} - \mathcal{R}_{header}}{\mathcal{R}_{min}}. \quad (5)$$

By evaluating the steady state probabilities numerically as explained in [4], the expected number of active interleaved frames is obtained as

$$k'_{avg} = \sum_{i=1}^{k'} i \times P_i^{Mode2}. \quad (6)$$

where P_i^{Mode2} denotes the probability that i users are assigned to one Mode-2 slot.

2.2 Quality of Service

We define the *quality of service* function, f_{QoS} , as the product of the number of users (U) in the system per slot per frame and the system throughput (η_{frame}). Thus,

$$\begin{aligned} f_{QoS} &= \left(\frac{U}{N} \right) * \eta_{frame} \\ &= \left[f_N \left(\frac{k_{avg}(1 - \hat{h}(k(1 + R) + 1))}{k(1 + R) + 1} \right) + (1 - f_N)(1 - \hat{h})(1 - p_l) \left(\frac{k'_{avg}}{k'} \right) \right] \times \\ &\quad \left[f_N k_{avg} + (1 - f_N) \left(\frac{k'_{avg}}{k'} \right) \right] \end{aligned} \quad (7)$$

For details, see [4]. In this formulation, η_{Mode1} and η_{Mode2} denote the throughput of a Mode-1 and Mode-2 slot respectively. In the above formulation the only parameter that can be tuned for an optimal performance is $f_N = \frac{N_1}{N_1 + N_2}$. Differentiating Equation (7) w.r.t. f_N leads to

$$f_N = - \left[\frac{k'_{avg}(\eta_{Mode1} - \eta_{Mode2}) + (k_{avg} * k' - k'_{avg})\eta_{Mode2}}{2(\eta_{Mode1} - \eta_{Mode2})(k_{avg} * k' - k'_{avg})} \right] \quad (8)$$

for which the function f_{QoS} is optimal.

To maximize f_{QoS} , we take the second derivative of Equation (7) w.r.t. f_N thus yielding the following condition

$$C1 : 2 * (\eta_{Mode1} - \eta_{Mode2})(k_{avg} * k' - k'_{avg}) \leq 0. \quad (9)$$

Recalling that f_N satisfies $0 \leq f_N \leq 1$, from condition $C1$ we derive two other conditions as follows.

$$C2 : \frac{\eta_{Mode1}}{\eta_{Mode2}} \leq 2 - \frac{k' * k_{avg}}{k'_{avg}} \quad (10)$$

$$C3 : \frac{\eta_{Mode1}}{\eta_{Mode2}} \geq \frac{4k'_{avg} - 3k' * k_{avg}}{3k'_{avg} - 2k' * k_{avg}} \quad (11)$$

Since k, k' are fixed based upon the data rate of the requests, k_{avg} and k'_{avg} are also fixed for a constant service rate of the slot (μ). Similarly p_l and \hat{h} are fixed. The restriction on the value of f_N gives us a range of values for μ and ω ($\omega = 1 - \mu$) as described in [4]. However, we do not yet have closed-form solutions for the values of k_{avg} and k'_{avg} which would give us the solutions for μ and ω . Therefore, we obtain the feasible values of μ and ω by simulation.

3 Proposed Slot Assignment Algorithm

3.1 A New Channel Assignment Scheme

- 1) From Equation (3) get the number of minislots to be allocated to users of each Mode-1 slot.
- 2) From Equation (5) get the number of interleaved frames for each Mode-2 slot.
- 3) From Equation (8) get the fraction of slots to be allocated to Mode-1 requests.
- 4) Allocate requests to Mode-1 and Mode-2 slots according to linear programming (LP) formulations (Sections 3.2 and 3.3).
- 5) If a packet is lost for a Mode-1 slot, a user appends the packet in the retransmission list. For a Mode-2 slot, a user retransmits the packet at the next opportunity.
- 6) On completion of a Mode-1 request, verify if that is the last job in that time slot. If so, call the linear programming routine for Mode-1 jobs.
- 7) On completion of a Mode-2 request, verify if it is the last job in that timeslot. If so, call the linear programming routine for Mode-2 jobs. Otherwise look for an identical job request waiting to be served; if successful, allocate the new job otherwise the slot in that frame remains idle.
- 8) If the airlink frame loss probability (p_l) changes by a fraction $\pm F$, we compute k and f_N dynamically. This calls for switching the existing calls from one slot to another slot. For a new value of p_l , the updated values are denoted by k_{new} and $N_{1,new}$ such that $N_{2,new} = N - N_{1,new}$. If $N_{1,new} > N_1$ then $(N_{1,new} - N_1)$ number of Mode-2 slots are allocated to Mode-1 users, otherwise $(N_1 - N_{1,new})$ number of Mode-1 slots are allocated to Mode-2 users. A greedy algorithm is used to adjust the users in slots which are in use. The slots with the largest amount of under-utilized bandwidth are chosen to allocate to the new traffic requests while the current users of those slots are adjusted in other slots used by the same traffic class users.

3.2 LP Formulation for Mode-1 Slot

Let d_1 be the number of possible data rates that can be expected from Mode-1 traffic. These rates are denoted as X_i^{Mode1} where $1 \leq i \leq d_1$. Let the number of requests waiting to be assigned to these data rates be denoted by N_i^{Mode1} and let the number of jobs of each data rate assigned to every slot be denoted by A_i^{Mode1} where $1 \leq i \leq d_1$. Since the number of minislots assigned to users in each slot is known, the available data bandwidth is $Available_{Mode1} = k\mathcal{R}_{min}$ which is nothing but the bandwidth available to transmit data after the header information for all users in that slot is transmitted. In order to minimize the amount of data bandwidth wasted, we find an optimal combination of user requests such that the bandwidth required is very close to the available bandwidth. The linear programming formulation for determining the combination of data rates for one slot allocation for Mode-1 jobs is given by

Minimize $\mathcal{F} \geq 0$; subject to the following constraints

$$\begin{aligned}\mathcal{F} &= k - \sum_{i=1}^{d_1} A_i^{Mode1}; Available_{Mode1} - \sum_{i=1}^{d_1} X_i^{Mode1} A_i^{Mode1} \geq 0; \\ A_i^{Mode1} &\leq N_i^{Mode1} \text{ and } A_i^{Mode1} \geq 0 \text{ for } 1 \leq i \leq d_1.\end{aligned}$$

3.3 LP Formulation for Mode-2 Slot

The formulation for Mode-2 slot is similar except that the available bandwidth for transmission by a user is $Available_{Mode2} = \mathcal{R}_{slot} - \mathcal{R}_{header}$. This is because the complete slot bandwidth corresponding to each frame is occupied by one user only. For Mode-2 jobs, let d_2 denote the number of possible data rates which are denoted as X_i^{Mode2} . Let the number of job requests for each data rate waiting to be allocated be N_i^{Mode2} , and let the number of jobs of each data rate assigned to a slot be denoted as A_i^{Mode2} where $1 \leq i \leq d_2$. The LP formulation is given by

Minimize $\mathcal{F} \geq 0$; subject to the following constraints

$$\begin{aligned}\mathcal{F} &= Available_{Mode2} - \sum_{i=1}^{d_2} A_i^{Mode2} X_i^{Mode2}, \\ A_i^{Mode2} &\leq N_i^{Mode2} \text{ and } A_i^{Mode2} \geq 0 \text{ for } 1 \leq i \leq d_2.\end{aligned}$$

4 Simulation Experiments

To simulate the proposed slot assignment algorithm, we considered six data rates of traffic for both Mode-1 and Mode-2 class of requests. These data rates are 2 Kbps, 4 Kbps, 8 Kbps, 16 Kbps, 24 Kbps, and 32 Kbps. We generated a frame with $N = 50$ slots with the frame bandwidth $N * \mathcal{R}_{slot} = 2$ Mbps which thus yields a slot bandwidth of $\mathcal{R}_{slot} = 40$ Kbps. Now $\mathcal{R}_{header} = 40 * \hat{h}$ where $\hat{h} = \frac{h}{D}$. The simulation results obtained from our scheme are compared with the performance of IS-136 protocol. Since IS-136 has only 6 slots per frame, the comparison is on a slot basis rather than on a frame basis. The simulation was carried out for different values of \hat{h} and p_l . The value of F (fractional change in p_l) was fixed to 0.05. We also assume that the activity of each user is 0.5 throughout the simulation. Hence the number of users assigned to each slot in the IS-136 protocol is 0.5.

In the absence of a closed form solution for f_{QoS} in terms of μ , the simulation was done for varying values of μ so as to satisfy the conditions $C1$, $C2$ and $C3$ as in Equations (9), (10), (11) respectively.

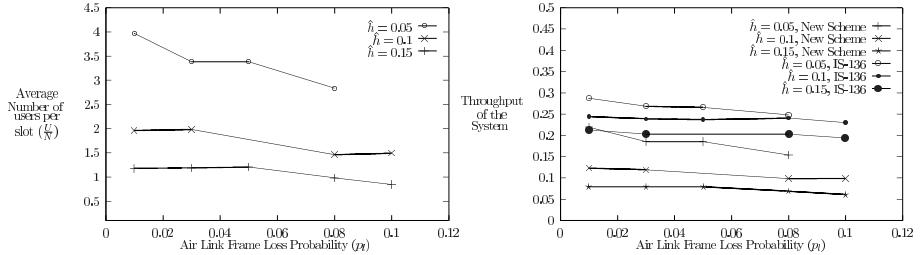
(a) $\frac{U}{N}$ versus p_l and \hat{h} (b) Throughput versus p_l and \hat{h}

Figure 3(a) plots the number of users served by each slot in each frame according to our proposed algorithm. As the overheads (header bits) increase, there is a drop in the capacity (number of users served). Similarly, there is a drop in the number of users served with increase in p_l , the airlink frame loss probability. According to the IS-136 protocol, activity of each user is 0.5 which remains constant. The maximum improvement in the capacity is as high as 800% and the minimum improvement is 75%.

Figure 3(b) plots the throughput of the system for varying values of p_l and \hat{h} . There is a drop in the throughput with increase in \hat{h} and p_l . The maximum degradation in the system throughput due to our proposed scheme is 67% and the minimum degradation is 25%. Inspite of the degradation in the throughput, the proposed scheme beats IS-136 in the overall system performance given by the f_{QoS} function.

5 Conclusions

We modeled a wireless system taking into consideration the *QoS* parameters. By an optimal utilization of the bandwidth we not only support a wide range of applications but also optimize the system performance. The proposed scheme performs better than IS-136 protocol when the airlink frame error rate and the overhead header bits are low. This clearly suggests that the existing slot allocation schemes need to be changed to suit user demands and network conditions.

References

1. J.-P. M.G. Linartz, "Packet-Switched Cellular Communication Architecture for IVHS using a Single Radio Channel", *Proc of IEEE PIMRC*, pp. 1222-1226, 1994.
2. T. Cheng and H. Tawfik, "Performance Evaluation of Two Channel Assignment Algorithms in Cellular Digital Packet Data Networks," *Proc of IEEE PIMRC*, pp. 537-543, 1995.
3. Naveen K. Kakani, S. K. Das, S. K. Sen and M. Kaippallimalil, "Optimizing QoS-based Channel Allocation in Wireless Data Packet Networks," *Proc of 7th IEEE Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD'98)*, Sao Paolo, Brazil, pp. 80-88, Aug 1998.
4. Naveen K. Kakani, S. K. Das, S. K. Sen, and M. Kaippallimalil, "A framework for Call Admission Control in Next Generation Wireless Networks," *Proc of First ACM International Workshop on Wireless Mobile Multimedia (WoWMoM '98)*, Dallas, pp. 101- 110, Oct 1998.

Harmony - A Framework for Providing Quality of Service in Wireless Mobile Computing Environment

Abhijit Lele and S.K. Nandy

Supercomputer Education and Research Center,
Indian Institute of Science, Bangalore - 560012 INDIA
`{abhijit,nandy}@serc.iisc.ernet.in`

Abstract. Recent advances in network and computer technology has lead to the integration of wireless and wireline network. In this paper we propose a complete framework called *Harmony* for providing quality of service over heterogeneous wireless and wireline networks. This framework helps allocate and manage both network and computational resources. The bandwidth is reserved based on the *Entropy* model. A *Static Load Mapping* and *Dynamic Load Balancing* scheme is proposed to allocate computational resources. Simulations are carried out to verify the correct functionality of the proposed model.

1 Introduction

Existing draft proposals such as UMTS [1] and IMT2000 [2] aim at providing mechanisms to guarantee quality of service (QoS) over a combination of wireless and wireline networks. Wide proliferation of personal communication systems have set up the stage for integrating wireline networks with wireless networks. The Ubiquitous communication project [3] aims at providing seamless interconnection for real time traffic over heterogeneous networks. The *Infopad* [4] project aims to provide low power hand held terminals, and the *Dataman* [5] project provides mechanisms to accommodate mobility in heterogeneous environment. Low power limits the compute capacity of hand held terminals and hence compute intensive applications need to be mapped to computational servers. Thus there is a need for a complete QoS architecture that can provide computational guarantees in addition to network guarantees. Several QoS architectures have been investigated in literature [8]. A dynamic QoS model referred to as QoS-A has been investigated by *Campbell et al* in [9]. Most of the models discussed in literature assume either a wireline or a wireless network. To the best of our knowledge, issues related to providing quality of service for heterogeneous networks is not discussed in any of the proposed architectures in literature. In this paper we propose a architecture called *Harmony* to provide compute and network guarantees over heterogeneous networks. A *Static Load Mapping* and *Dynamic Load Balancing* scheme is proposed to provide compute guarantees in a mobile computing environment. The rest of the paper is organized as follows. The *Harmony*

architecture and proposed service classes are discussed in section 2. Section 3 discusses the communication traffic models and the resource allocation scheme. The computational models and resource allocation scheme is discussed in section 4. The static load mapping and dynamic load balancing scheme is also discussed in this section. Section 5 is devoted to simulations and results. We finally conclude in section 6.

2 Framework for Supporting QoS

We propose the following hierarchy of classes shown in figure 1 which take into account the mobility and compute service requirement of incoming sessions. Any

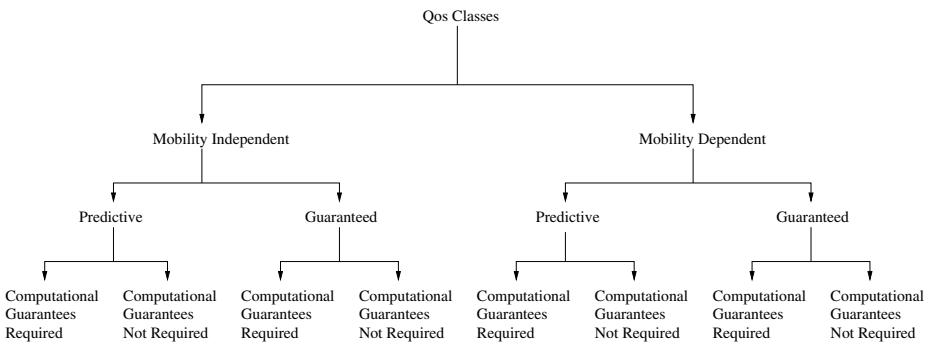


Fig. 1. QoS Class Structure

incoming session can subscribe to either *Mobility Independent* or *Mobility Dependent* service class with guaranteed or predictive service. Four classes of compute guarantees are defined in table 1 for sessions requesting for computational guarantees. The classification into classes of different compute requirement is

Table 1. Table of Association of Class with Compute Requirement

Compute Guarantee Class	I	II	III	IV
Compute Requirement (MFLOPS)	50	100	200	400

heuristic in nature and more work needs to be done to arrive at an exact classification.

Low power constraints of mobile hand held terminals limit their compute capacity. Thus compute intensive applications need to be mapped onto a compute server. A two level hierarchical network called as *Mobile Computing Environment*

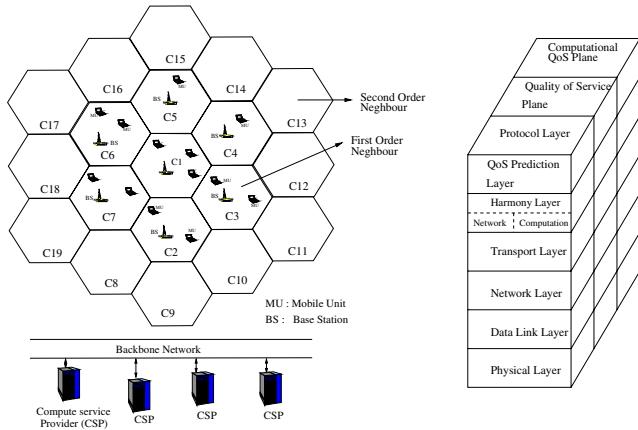


Fig. 2. Mobile Computing Environment and Harmony Architecture

(MCE) shown in figure 2 provides such an environment. The base stations(BS) which provide wireless connectivity to mobile users are connected over a wireline network to form the first level of hierarchy. *Compute Service Providers (CSP)* are associated with a set of base stations and form the second level of hierarchy. These CSP's serve as computational units to which computations from mobile users are mapped. The CSP's also form the interface to external world over B-ISDN network.

We propose a layered QoS architecture called as *Harmony* shown in figure 2 that meets the QoS principles in [9]. In functional terms harmony consists of number of layers and planes. The QoS prediction layer comprises of the network and computation resource prediction mechanisms. The network resource prediction is based on the *entropy* model proposed in [6], and the computational model is based on the class of service requested by the user. Resource allocation and call admission control mechanisms are implemented in the harmony layer. The computational resources are reserved based on the mobility model called as *Static Load Mapping* and *Dynamic Load Balancing* which are implemented within the harmony layer. The vertical planes are used for QoS management after a session has been established. The QoS plane consists of a number of layer specific QoS managers. The compute QoS plane helps maintain compute guarantees throughout the duration of the session.

3 Communication Traffic and Resource Allocation

Communication resources primarily comprise of bandwidth and buffer space. In order to reserve these resources in a heterogeneous environment, network traffic profile and mobility profile of users must be known *a priori* [7] [10]. We extend the *Entropy* based model proposed in [6] to determine network traffic profile

over wireless channels. A synthetic traffic profile, having the same statistical properties as that of traffic in wireless environment, is generated by modulating traffic profile in a wireline environment with white Gaussian noise. This synthetic stream is modeled using the *Entropy* model with 21 states. Network resources are reserved based on the bandwidth requirement function as given in [6]. The network traffic model forms a part of the QoS prediction plane in the framework given in figure 2. The guaranteed delay schedule and the guaranteed bandwidth schedule [6] form a part of the vertical QoS plane. Having determined the network traffic profile, the resources can be reserved according to the following scheme. Consider the cellular network comprising of hexagonal shaped cells [11] shown in figure 2. We define

Definition 1. First Order Neighbors (FON) : *A cell C_i is called as the FON of cell C_j if C_i has one boundary common with C_j .*

Definition 2. Second Order Neighbor (SON) : *A cell C_i is called as the SON of cell C_j if there exists a cell C_k which is a FON of C_i and C_j is the FON of C_k .*

Let a mobile unit M_i originate a call in cell C_i requesting for bandwidth B . Let F_i and S_i be the set of cells that constitute the FON's and SON's of cell C_i respectively. Let α and β be fractions such that $0 \leq \alpha, \beta \leq 1$. The call admission policy and resource allocation policy for various classes of traffic are given in table 2. In mobility independent predictive and mobility dependent predictive class of traffic, bandwidth allotted in F_i and S_i can be used by other sessions subscribing to predictive services. These resources can be reclaimed back from the user for whom the resources are reserved. Note that we permit sharing

Table 2. CAC and Resource allocation policy for various traffic classes

Traffic Class	Minimum bandwidth necessary in cells C_i, F_i, S_i for a call to be admitted			Amount of resources reserved in cells C_i, F_i, S_i on call admission		
	C_i	F_i	S_i	C_i	F_i	S_i
Mobile Independent Predictive	B	B	B	B	B	B
Mobile Independent Guaranteed	B	B	B	B	B	B
Mobile Dependent Predictive	B	$\alpha \times B$		B	$\alpha \times B$	
Mobile Dependent Guaranteed	B	$\alpha \times B$	$\beta \times B$	B	$\alpha \times B$	$\beta \times B$

network resources reserved by one session by other sessions in certain class of traffic to improve network resource utilization. The harmony layer which works in co-operative collaboration with the QoS plane is responsible for reserving communication resources.

4 Computation Models and Resource Allocation

It is difficult to generalize computational requirements for various traffic classes. On the other hand, rule of thumb may be applied to estimate the computation necessary based on the type of application. For sessions in which applications are not known *a priori*, an average quanta of computational resources is reserved. This is dynamically changed as and when the application type is known. Such a scheme can be provided only in predictive class of services. Guaranteed class of service must specify its computational requirement based on the classification given in table 1.

We propose the following scheme to reserve compute resources. Consider the MCE shown in figure 2. Let $\langle P_1 \dots P_n \rangle$ denote the CSP's associated with base stations $\langle B_1 \dots B_m \rangle$. In particular let P_i and P_j be the CSP's associated with base station B_i and B_j respectively. We assume that when a handoff of a mobile user occurs from base station B_i to base station B_j , the associated computations are also handed over from CSP P_i to CSP P_j . Every incoming session requests for certain computational resources during call establishment. Let C_{max} denote the total computational requirement requested by the mobile user M_i originating a call from base station B_i . We propose the following scheme to guarantee computational QoS.

1. **Mobility Independent** : A call is admitted when C_{max} can be allotted to the session on all the computational units $\langle P_1 \dots P_n \rangle$. Computational resources equivalent to C_{max} are reserved on all the CSP's $\langle P_1 \dots P_n \rangle$.
2. **Mobility Dependent** : A call is admitted when C_{max} can be split into subset of computational requirement $\langle c_1 \dots c_k \rangle$ such that

$$C_{max} = \sum_{i=1}^{i=k} c_i \quad (1)$$

and we can find set of K CSP's such that c_i can be guaranteed over P_i . Computational resources equivalent to c_i are reserved over P_i . We refer to this scheme as *Static Load Mapping (SLM)*. We shall discuss in detail about SLM in the following section.

4.1 Static Load Mapping

We propose a *Static Load Mapping* scheme to provide computational guarantees taking into account the mobility of users. Consider the cellular network shown in figure 2. Let a mobile unit M_i originate a call in base station B_i , request for a computational guarantee of C_{max} . The value of C_{max} depends on the class of service given in table 1. Let (x_0, y_0) be the initial position of the mobile user and let v be the associated velocity of M_i . Let $P_1 \dots P_n$ be the set of CSP's having computational capacity $\omega_1 \dots \omega_n$. Knowing the cell dimensions, the mobile user M_i remains in the cell for a time $t_i = \frac{v}{d}$, where d is the distance from (x_0, y_0) to the cell boundary. Thus the mobile user M_i traverses along the path

$$(x_0, y_0) \xrightarrow{\frac{v}{d}} (x_1, y_1) \xrightarrow{\frac{v}{d}} \dots (x_k, y_k)$$

which forms the mobility profile of the mobile user. The mobile user enters another cell after a time t_i and hence a mobile unit is associated with a CSP P_i for a duration of t_i . Let $P_1 \dots P_n$ be the CSP's along this mobility profile. Thus the maximum computations that can be carried out on P_i is $\omega_i \times t_i$. Set $c_i = \omega_i \times t_i$ and reserve this amount of compute resource on P_i . Note by using this scheme we are guaranteeing QoS at static time taking into account the mobility of the user. An optimal utilization of computational resources can be achieved through dynamic load balancing discussed in the following section.

4.2 Dynamic Load Balancing

In this paper we propose a *Dynamic Load Balancing* scheme to optimally allocate computational resources. As defined earlier a mobile unit remains in a cell for a maximum duration of t_i (deadline t_i) in which c_i work has to be performed. Let δt denote the actual time taken to complete the work. It therefore follows that $\delta t \leq t_i$ where $\delta t = \frac{c_i}{\omega_i}$. In the dynamic load balancing scheme we associate a queue Q_i with every P_i . The entries in the queue are the work $< c_1 \dots c_m >$ that need to meet deadline $< t_1 \dots t_m >$ respectively. We define the following terms

Definition 3. If δ_i is the time taken to complete the work c_i , the **Actual Work Load** W_a is

$$W_a = \sum_{j=1}^{j=m} \omega_i \times \delta_i \quad (2)$$

where m is the total number of active sessions mapped onto processor i .

Definition 4. Tolerable Work Load of processor P_i over a duration T is

$$W_T = \omega_i \times T \quad (3)$$

We define *High Water Mark (HWM)* to be ρ times the tolerable work load and *Low Water Mark (LWM)* to be η times of the tolerable work load, where $0 \leq \rho, \eta \leq 1$. The values of ρ and η are selected depending on traffic classes. Let an incoming session k request for work load Ω which is split into subtasks having work load $w_1^k \dots w_m^k$. This work load needs to be mapped to processors $P_1 \dots P_m$. If W_a^i is the actual work load on processor i , a call is admitted with computational guarantees only when $W_a^i + w_i^k \leq W_T \forall i$, and a mapping is possible only when $W_a^i + w_i^k \leq HWM \forall i$. If for some processor P_j , $W_a^j \leq LWM$, then we always map a W_i^k to P_j to balance the load. Work load is migrated across processors whenever load in any processor exceeds its HWM to maintain a load balance. This computational call admission and resource allocation policy is mapped on to the computational guarantee plane in the harmony architecture.

5 Simulations and Results

Simulation were called out for 64 cell micro cellular environment with cell diameter assumed to be less than one kilometer. A CSP is associated with a group of four cell and hence there are *sixteen* CSP's each having the same compute capacity. A handoff of CSP occurs only after the mobile unit crosses four cells. The wireless access scheme is assumed to be TDMA with equal number of channels per base station and we use dynamic channel allocation policy as described in [11]. Let λ denote the average traffic load per cell. The duration of every mobile originated call is assumed to be independent exponentially distributed random variable with a average duration of 5 minutes. The maximum velocity (Km/hour) of a mobile user is a uniform random variable between 0 and 20. Also the direction (degrees) of the mobile unit is a uniform random variable between 0 and 360. Throughout this simulation we assume the mobile unit moves with a constant velocity. We also assume that all the incoming sessions request for real time multimedia connection. The simulations were carried out for more than 10000 calls per cell. The metrics used in the simulation were *Percentage Call Blocking*, *Network Resource Utilization* and *Compute Resource Utilization* [11]. Let P_{ind} , P_{dep} , P_{gur} , P_{pred} denote the probability of mobile independent, mobility dependent, QoS guarantees required, and predictive QoS guarantees required class of traffic. Simulation were carried out for the scenarios given in table 3 and the results obtained are given in table 4 and figure 3. The call blocking graph plots traffic load (x-axis) vs percentage call blocking (y-axis). The network utilization graph plots traffic load (x-axis) vs percentage network utilization (y-axis) and the compute utilization graphs plots traffic load (x-axis) vs percentage compute resource utilization (y-axis).

Table 3. Scenarios for Simulation

Scenario	Traffic Load λ	Probability of Traffic Class
Scenario 1	$0.1 \leq \lambda \leq 0.5$	$P_{ind} = P_{dep}$ and $P_{gur} = P_{pred}$
Scenario 2	$0.1 \leq \lambda \leq 0.5$	$P_{ind} \leq P_{dep}$ and $P_{gur} = P_{pred}$
Scenario 3	$0.1 \leq \lambda \leq 0.5$	$P_{ind} \geq P_{dep}$ and $P_{gur} = P_{pred}$
Scenario 4	$0.1 \leq \lambda \leq 0.5$	$P_{ind} = P_{dep}$ and $P_{gur} \geq P_{pred}$
Scenario 5	$0.1 \leq \lambda \leq 0.5$	$P_{ind} = P_{dep}$ and $P_{gur} \leq P_{pred}$

Simulations indicate the dependency of call blocking probability on α, β and λ . Increasing α and β reduces the network utilization. The number of sessions permitted to enter the network increases when predictive class of traffic is more probable than guaranteed class of traffic. Simulations for $\alpha \geq \beta$ indicate that the general trend is the same as that of the results presented in this paper. If $\alpha \leq \beta$, then the average network utilization decreases as compared to $\alpha \geq \beta$. Also note that in the all the five scenarios simulated the number of calls dropped

Table 4. Simulation results for different Scenarios

Traffic Load	Percentage of predictive calls dropped due to unavailability of		Percentage of guaranteed calls dropped due to unavailability of		Percentage of calls Dropped due to handoff	
	Compute Resource	Network Resource	Compute Resource	Network Resource	Compute Resource	Network Resource
Scenario 1						
0.1	1.27	0.51	1.18	0.60	0.12	0.17
0.2	1.37	0.71	1.08	1.01	0.25	0.29
0.3	1.49	0.97	1.30	1.15	0.37	0.46
0.4	1.82	1.12	1.63	1.31	0.41	0.37
0.5	2.03	1.35	2.95	1.43	0.67	0.75
Scenario 2						
0.1	0.41	0.62	0.47	0.56	0.11	0.13
0.2	0.68	0.56	0.59	0.65	0.17	0.21
0.3	0.81	0.94	0.88	0.87	0.23	0.33
0.4	0.63	0.74	1.67	1.51	0.44	0.41
0.5	0.77	0.76	1.78	1.58	0.77	0.55
Scenario 3						
0.1	1.10	1.00	1.78	1.81	0.37	0.61
0.2	0.72	1.18	1.91	2.13	0.41	0.58
0.3	0.86	0.93	2.03	2.28	0.60	0.55
0.4	1.32	1.46	1.87	2.06	0.73	0.73
0.5	1.47	1.51	2.69	2.74	0.85	0.79
Scenario 4						
0.1	3.26	2.68	6.87	7.01	1.64	1.82
0.2	3.11	3.22	7.41	7.38	1.72	1.93
0.3	3.97	2.89	7.82	8.2	2.15	2.17
0.4	4.45	3.66	9.74	9.25	2.68	2.88
0.5	4.83	4.21	10.07	11.05	3.18	3.46
Scenario 5						
0.1	1.93	2.15	0.83	0.91	0.73	0.31
0.2	2.14	2.64	0.95	1.10	0.46	0.39
0.3	2.66	2.72	1.26	1.04	0.53	0.58
0.4	3.46	3.02	1.46	1.29	0.77	0.91
0.5	4.28	4.09	1.73	.187	0.83	0.80

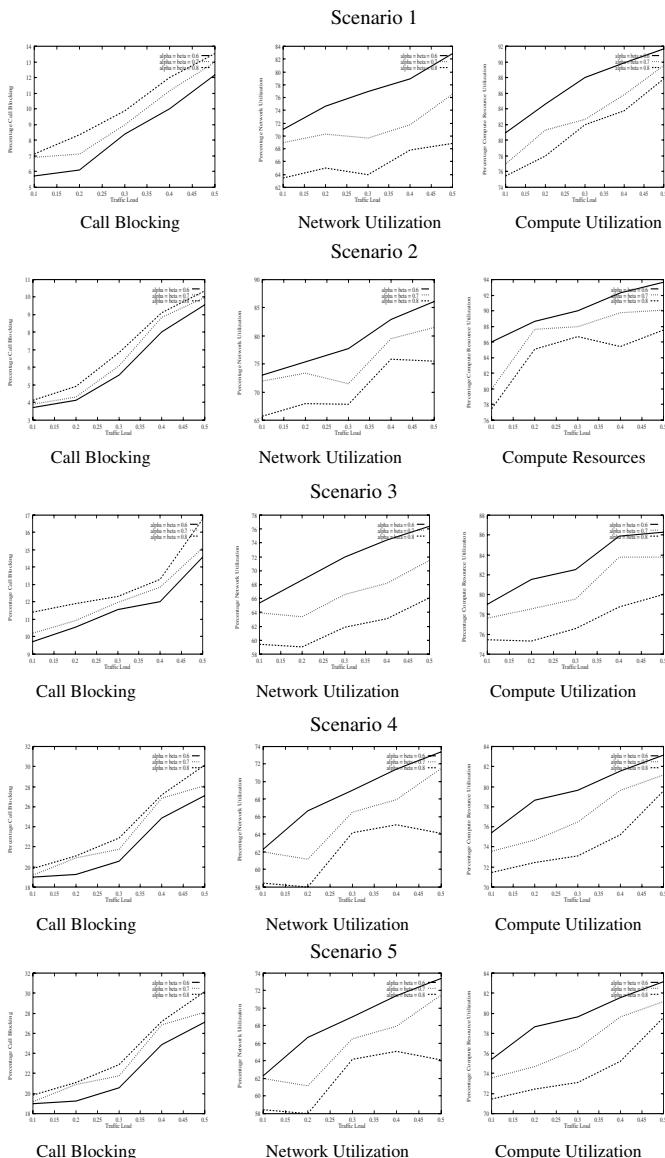


Fig. 3. Simulation plots

during handoff is very low indicating that the QoS is indeed being guaranteed throughout the duration of the call.

6 Conclusion

With the integration of wireless and wireline networks there is a need to provide mechanisms to guarantee quality of service for real time traffic over such heterogeneous networks. In this paper we have proposed a complete framework called as *Harmony* to guarantee quality of service over mobile computing environments. The proposed harmony model is a layered architecture and each layer work in co-operative collaboration with other layers to provide QoS guarantees. Eight classes of traffic have been proposed to provide network and computational guarantees. The bandwidth requirement for any incoming session are predicted based on the *Entropy* model. A *Static Load Mapping* and *Dynamic Load Balancing* schemes are proposed to provide compute guarantees in a mobile environment. Simulations are carried out for various scenarios. From the simulations we find that network and computational resource utilization is maximum when predictive class of traffic dominates, and the network and compute resource utilization is minimum when guaranteed class of traffic dominates.

References

1. Universal Mobile Telecommunication System, Technical Reports, <http://umts-forum.org/reports.html>
2. International Mobile Telecommunications : IMT-2000, Specifications and Documents <http://www.itu.int/imt/1-info/docs/information-docs/index.html>
3. Ubiquitous Communications Project at Technical University, Delft, Technical Report <http://www.ubicom.tudelft.nl/projects>
4. Infopad Project at University of California, Berkeley, White Paper, http://infopad.eecs.berkeley.edu/ pering/infopad_whitepaper.html
5. Dataman Project at Rutgers University, <http://www.cs.rutgers.edu/ badi/dataman>
6. Abhijit Lele and S.K. Nandy, "Can QoS Guarantees be Supported for Live Video Over ATM Networks", *Proc. IEEE Conference on Global Communications*, Nov. 1998.
7. Roy Mauger and Catherine Rosenberg, "QoS Guarantees for Multimedia Services on TDMA based Satellite Network", *IEEE Communications Magazine*, July 1997, pp. 56-65.
8. Cristina Aurrecoechea, A.T. Campbell and Linda Hauw, "Survey of QoS Architectures", *Tech. Report, Center for Telecommunication Research, Columbia University*, (<http://www.ctr.columbia.edu/comet/members.html>)
9. Andrew Campbell *et al*, "Supporting Adaptive Flows in Quality of Service Architectures" *ACM Multimedia Systems Journal*, 1996.
10. Harry G. Perros and Khaled M. Elsayed, "Call Admission Control Schemes a Review", *Technical Report, North Carolina State University*. (http://www.ece.ncsu.edu/cacc/tech_reports).
11. Y.C. Lee, "Cellular Communications", *McGraw Hill*, 1996.

Stochastic Modeling of TCP/IP over Random Loss Channels

Alhussein A. Abouzeid, Murat Azizoglu, and Sumit Roy*

Department of Electrical Engineering, University of Washington,
Box 352500
Seattle, WA 98195-2500

Abstract. An analytical framework for modeling the performance of a single TCP session in the presence of random packet loss is presented that is based on a semi-Markov model for the window size evolution. The model predicts the throughput for LANs/WANs (low and high bandwidth-delay products) with good accuracy, as compared against simulation results with *ns* simulator. Generally, higher speed channels are found to be more vulnerable to random loss than slower channels, especially for moderate to high loss rates.

1 Introduction

TCP/IP has been designed for reliable (wired) networks in which packet losses occur primarily due to network congestion. TCP employs window-based end-to-end congestion avoidance [6] by sending an acknowledgment (ACK) back to the source for each successful packet. At all times, the source keeps a record of the number of unacknowledged packets that it has released into the network, called the *the window size*. The source detects a packet loss by either the non-arrival of a packet ACK within a certain time (maintained by a *timer*), or by the arrival of multiple ACKs with the same next expected packet number. A packet loss is interpreted as an indication of congestion, and the source responds by reducing its window size so as not to overload the network with packets. Modeling this dynamic behavior of congestion window size is key to analyzing TCP/IP throughput performance.

In some circumstances (e.g. networks with wireless links), packet losses occur randomly due to link effects than due to network congestion. While random packet loss on the Internet has been reported in [7], it was not taken into consideration in the design of TCP/IP congestion control. Previous research [2,3,4] has shown that random packet loss (which is not due to congestion) may severely decrease the throughput of TCP because TCP interprets random packet loss to be due to congestion and hence lowers the input data rate into the network, and consequently the throughput. In [2,3], a discrete-time model for random packet loss was used in which any given packet is lost with probability q independent

* Author for all correspondence - Ph/FAX : (206) 221-5261/543-3842; e-mail:
`roy@ee.washington.edu`

of all other packets implying a geometric distribution on the number of successful packets between consecutive loss events. In [4], packet loss is characterized by an inhomogeneous Poisson process and the steady-state distribution of the window size obtained under the assumption of infinite buffer size. In this work, we assume a continuous-time packet loss model governed by a general renewal process and incorporate (finite) buffer sizing impact on TCP performance.

Our basic system model assumes an infinite source that releases packets into a buffer of size B upon receiving ACKs from the destination. Packets are sent over a link with capacity μ packets/second and a net delay of τ (propagation delay plus any other processing delays etc.). Define $T = \tau + 1/\mu$ to be the time between the start of transmission of a packet and the reception of an ACK for this packet. Then μT is the bandwidth-delay product and the ratio $\beta = \frac{B}{\mu T}$ is the buffer size normalized by the bandwidth-delay product.

2 Ideal Channels without Random Packet Loss

We first briefly review the operation of TCP-Reno (TCP-R) for ideal channels, and summarize the key results in [1,3].

Denote $w_p = \mu T + B = \mu \tau + B + 1$, and note that when the window size reaches w_p , the bit pipe (the combination of the channel and the transmit buffer) is fully utilized. A further increase in window size at this stage causes buffer overflow, at which point the window size is halved and W_{th} is set to $w_p/2$. Let $t' = 0$ denote the time of establishment of the TCP session under consideration, and let $W(t')$ denote the congestion window size at time t' . Let n denote the number of packets acknowledged during a time interval t . The *deterministic* window size $W(t')$ evolution during a TCP session has been analyzed in [1,3] and yield useful expressions that are summarized below (see Figure 1 and the original sources for details).

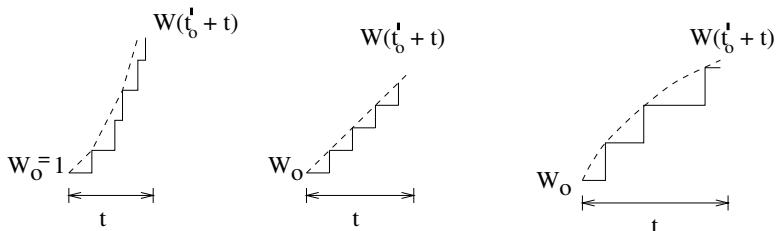


Fig. 1. Sketch of the exponential, linear and sub-linear $O(\sqrt{t})$ phases for window evolution. Solid lines indicate the actual window size evolution while dotted lines indicate the envelope.

1. **Slow Start** ($1 < W(t') < W_{th}$). Consider two instants $t'_0, t'_0 + t$ in a slow start phase of any of the TCP cycles. Choose t'_0 such that $W(t'_0) = 1$. Then,

$$W(t'_0 + t) = 2^{t/T} \quad (1)$$

$$n = W(t'_0 + t) - 1 \quad (2)$$

2. **Congestion Avoidance - Phase I** ($W_{th} < W(t') < \mu T$). Consider two instants $t'_0, t'_0 + t$ in a congestion avoidance phase of any of the TCP cycles. Choose t'_0 such that $W(t'_0) = W_0$. Then,

$$W(t'_0 + t) = W_0 + t/T \quad (3)$$

$$n = \frac{1}{T}(W_0 t + t^2/(2T)) \quad (4)$$

3. **Congestion Avoidance - Phase II** ($\mu T < W(t') < w_p$). Consider two instants $t'_0, t'_0 + t$ in a congestion avoidance phase of any of the TCP cycles. Choose t'_0 such that $W(t'_0) = W_0$. Then,

$$W(t'_0 + t) = \sqrt{W_0^2 + 2\mu t} \quad (5)$$

$$n = \mu t \quad (6)$$

It is apparent that the TCP window size evolution is periodic, i.e., consists of TCP ‘cycles’. Using (1) - (6), the average packet transmission rate R is the ratio of the number of packets sent in one cycle of the TCP session to the time duration of the cycle, i.e.,

$$\beta < 1 : R = \frac{n_A + n_B}{t_A + t_B} \quad (7)$$

$$\beta > 1 : R \simeq \mu \quad (8)$$

and the corresponding average throughput is

$$\rho = \frac{R}{\mu} \quad (9)$$

The values for n_A, n_B, t_A and t_B are obtained by substituting for W_0 and $W(t')$ (see Figure 1) in (3)-(6) by the initial and final values of the slow start and congestion avoidance phases. Note the difference in the expressions for $\beta < 1$ and $\beta > 1$ - for $\beta < 1$, the window size evolution contains a linear growth phase described by (3) during congestion avoidance, which doesn’t exist in the latter.

3 Channels with Random Packet Loss

3.1 Random Loss Model

Let S_i denote the time of the i^{th} packet loss, for $i = 1, 2, \dots$ and $X_i = S_i - S_{i-1}$ the time between $(i-1)^{\text{th}}$ and i^{th} loss events with $X_1 = S_1$. We consider

$\{X_1, X_2, \dots\}$ to be a set of IID random variables with probability density function $f(x)$ and distribution function $F(x)$. Thus, the process (pdf) defined by the loss occurrence times $\{S_1, S_2, \dots\}$ is a renewal process with inter renewal pdf $f(x)$.

Now, suppose that at a certain time instant $X_1 (=S_1)$, the first *random* packet loss event occurs. Denote the window size at that instant by W_1 . When the source detects this loss (by the arrival of duplicate ACKs for the case of TCP-R), the window size is halved. The window size now increases as depicted earlier (the window size starts from $W_1/2$ and increases till w_p , at which time a buffer overflow takes place and $W(t')$ is set to $w_p/2$, and so on) until another random packet loss takes place at a random time instant $S_2 = X_1 + X_2$. Denote the window size at this time (the time of the second loss) by W_2 .

In what follows, we call one period from $w_p/2$ till w_p the free-running period or the ‘typical’ cycle (i.e. free from random loss effects). Note that the second random loss event can happen before the occurrence of any ‘typical’ cycles. The window size $W(t')$ is a semi-Markovian stochastic process, because the window size evolution after a random loss (except for its starting value which is half of that just before the random loss) is statistically independent from the window size evolution before the random loss. Further, since $\{X_1, X_2, \dots\}$ are independent and identically distributed (IID), the window sizes $\{W_1, W_2, \dots\}$ (window sizes just before the random loss) form a finite state Markov Chain (i.e. the embedded Markov Chain of the semi-Markov process $W(t')$) [8].

3.2 Analysis

For the above model, we wish to compute the following quantities for the embedded Markov Chain

(1) $E[N|W_1 = w_1]$, the expected number of packets successfully transmitted before another random packet loss occurs, given that the most recent random loss took place at w_1 ; (2) The conditional probability $P[W_2 = w_2|W_1 = w_1]$ (denoted for convenience by P ; the probability that the next *random* loss takes place at $W_2 = w_2$ given that the previous *random* loss took place at $W_1 = w_1$.

To do this, we will ignore the first cycle of TCP-R and assume that the TCP session starts with window size $w_p/2$ (instead of 1) - this approximation should have a negligible effect on the average throughput since (i) a source with an infinite number of packets was assumed, hence the transient behavior (slow start) at the beginning of the connection is expected to be negligible, even for the case of random loss; and (ii) the duration as well as the number of packets sent during this slow start phase is small.

Two ranges of β are considered separately, $\beta < 1$ and $\beta > 1$, and expressions for $E[N|W_1]$ and $P[W_2|W_1]$ are found for each of the two ranges.

Define,

N_a, N_b : the number of packet transmissions during Congestion Avoidance Phase I and II, respectively, of the atypical cycle following a random packet loss at a window size $W_1 = w_1$.

N_A, N_B : the number of packet transmissions during Congestion Avoidance Phase

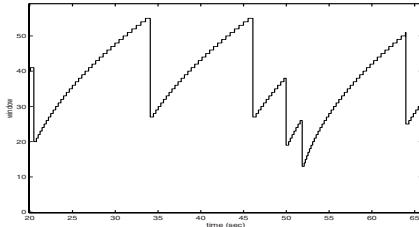


Fig. 2. Sample function of the window size evolution with random packet loss for a TCP-R session. $\mu = 100$, $\tau = 0.1$, $\beta = 4.0$ and $E[X] = 10$.

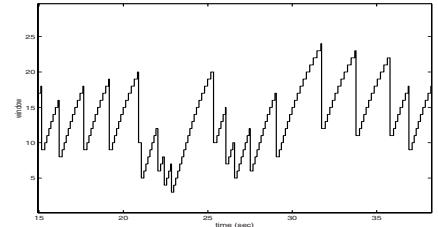


Fig. 3. Sample function of the window size evolution with random packet loss for a TCP-R session. $\mu = 100$, $\tau = 0.1$, $\beta = 4.0$ and $E[X] = 1$.

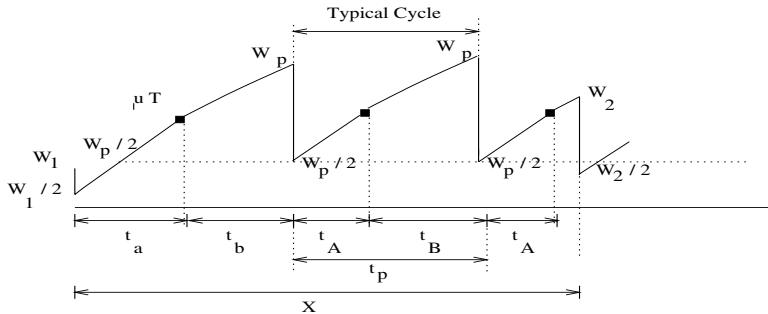


Fig. 4. A sketch of a sample function of window size with random loss for TCP-R ($\beta < 1$).

I and II, respectively, of a typical cycle.

$N_p = N_A + N_B$ is the number of packets sent in a typical cycle ($N_A = 0$ for $\beta > 1$).

The corresponding durations of time where the above number of packets is transmitted (time is counted since the beginning of the phase referenced) are t_a , t_b , t_A , and t_B respectively. Thus, $t_1 = t_a + t_b$, $t_p = t_A + t_B$ are the durations of the atypical and a typical cycle, respectively. For $\beta > 1$, $N_A = 0$ and $t_A = 0$.

Further details of the derivation and the results for a general inter-loss distribution $F_X(x)$ are contained in [10] and are ommitted due to space constraints. The analysis results for the case of $\beta < 1$ and $F_X(x) = e^{-\lambda x}$ are:

$$E[N|W_1 = w_1] = \sum_{n=0}^{na-1} e^{-\lambda(T/2)} \sqrt{w_1^2 + 8(n+1)} - w_1 + \frac{e^{-\lambda t_1}}{1 - e^{-\lambda t_p}} \sum_{n=0}^{nA-1} e^{-\lambda(T/2)} \sqrt{w_p^2 + 8(n+1)} - w_p \\ + e^{-\lambda/\mu} \frac{1 - e^{-(\lambda/\mu)nB}}{1 - e^{-(\lambda/\mu)}} (e^{-\lambda t_a} + \frac{e^{-\lambda t_1}}{1 - e^{-\lambda t_p}} e^{-\lambda t_A}) \quad (10)$$

$$P = \begin{cases} 0 & 0 < w_2 < w_1/2 \\ e^{-\lambda T(w_2-w_1/2)}(1-e^{-\lambda T}) & w_1/2 < w_2 < w_p/2 \\ e^{-\lambda T w_2}(1-e^{-\lambda T})(e^{\lambda T w_1/2} + \frac{e^{-\lambda(t_1-w_p/2)}}{1-e^{-\lambda t_p}}) & w_p/2 < w_2 < \mu T \\ e^{\lambda \frac{(\mu T)^2}{2\mu}}(e^{-\lambda w_2^2} - e^{-\lambda(w_2+1)^2})(e^{-\lambda t_a} + \frac{e^{-\lambda(t_1+t_A)}}{1-e^{-\lambda t_p}}) & \mu T < w_2 < w_p \end{cases} \quad (11)$$

Finally, the average packet transmission rate R is computed from

$$R = \frac{E[N]}{E[X]} \quad (12)$$

where $E[N]$ is the average number of packets successfully sent in an inter-loss duration, and $E[X]$ is the average time between two random losses ($= 1/\lambda$). $E[N]$ is given by

$$E[N] = \sum_{W=0}^{w_p} E[N|W]\pi(W) \quad (13)$$

where π is the steady state distribution of the MC. π is numerically computed using the eigensolver routines in MATLABTM for $P[W_2|W_1]$ for different values of λ , μ , τ and B .

4 Simulations Results and Concluding Remarks

In the packet level C code simulations, we considered the same set-up described in the system model in Section 1. The results from the analysis match closely the results from the simulations (Figure 5). Neglecting the slow start phase at the beginning of a TCP-R session in the analysis contributes in some deviation between the simulation and analysis results. For a given channel (i.e. bandwidth-delay

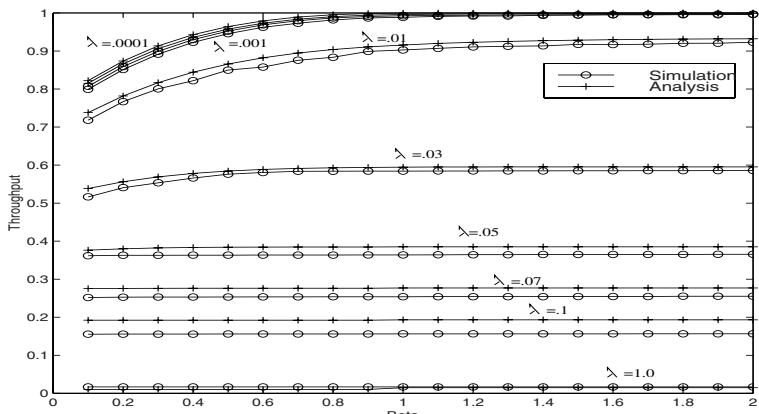


Fig. 5. $\mu = 100, \tau = 1.0$

product), the deviation (between the simulation and the analysis results) for low loss rates is small. This is because the slow start phase duration is sufficiently small such that the window size reaches $w_p/2$ in a very short time (compared to the average time to the first random loss) corroborating our approximation. In the analysis. As λ increases, so does the deviation since it becomes increasingly probable that the first random loss takes place early in the slow start phase, thereby precipitating a congestion avoidance phase with an initial window size that is considerably smaller than $\frac{w_p}{2}$ as assumed in the approximation. Consequently, the simulated throughput (on the average) is lower than that predicted by analysis, most noticeably for moderate values of random loss. For heavy loss rates, the deviation decreases again since the approximate window size quickly decreases from its starting value of $w_p/2$ to that (i.e., the true) in the simulations.

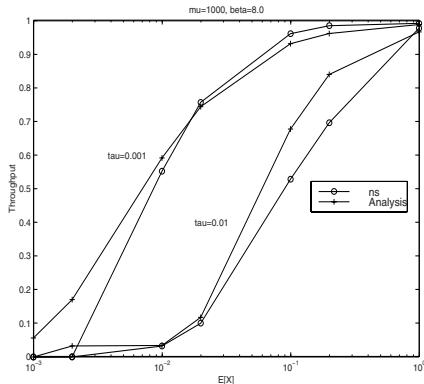


Fig. 6. Comparison between analysis and *ns* results for memoryless channels.

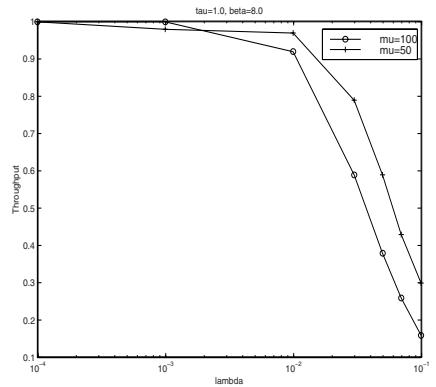


Fig. 7. Throughput comparison for two links with different speeds.

The analytical results based on the proposed random loss model matches with the *ns* simulation results as shown in Figure 6. The *ns* simulations are done using a two node topology and default TCP-R parameters (packet size = 1000 bytes, unlimited receiver's advertised window and $T_{c tick} = 0.01$).

The main conclusions that can be deduced from the throughput behavior in Figures 5, 6 and 7 are summarized below:

(1) For a link with a loss rate λ and a bandwidth-delay product $\mu\tau$, the results show that increasing the buffer size (i.e. increasing β) does *not* always increase the throughput. For channels with high loss rate, increasing the buffer size has no positive effect on the throughput; however for channels with low loss rates, increasing the buffer size increases the throughput considerably.

(2) For low loss rates, faster channels (higher μ) have higher throughput. However (contrary to what may be expected) for moderate to high loss rates, slower channels have higher throughput. The explanation for this is simple though perhaps not transparent. Recall that for channels without random loss,

the throughput is given by $\frac{n_p/t_p}{\mu}$. For channels with random loss, the throughput is given by $\frac{\lambda E[N]}{\mu}$. The expression in the numerator is the average transmission rate. Now, for the case of no random loss, increasing μ increases n_p significantly and hence the average transmission rate as well as the throughput increase. Similarly, for low random loss rates, increasing μ increases $E[N]$ significantly and hence both average rate and throughput increase. On the other hand, for moderate to high loss rates, increasing μ does not increase the number of packets successfully transmitted proportionately (due to the effect of random loss); hence the average transmission rate increases but the throughput actually decreases.

One practical interpretation of this result for the Internet relates to a user's dial-up modem connection to a server. Purchasing a faster modem would increase the average transmission rate, but may not be economically justifiable in the case of moderate-to-high loss rate channels since the proportion of the used bandwidth (i.e., throughput) for the new faster modem is less than that for the slower (and hence, less expensive) one.

References

1. S. Shenker, L. Zhang, and D. D. Clark, "Some observations on the dynamics of a congestion control algorithm," *Computer Communications Review*, pp. 30-39, Oct 1990.
2. T. J. Ott, J. H. B. Kemperman, and M. Mathis, "The stationary behavior of ideal TCP congestion avoidance," 1996.
3. T. V. Lakshman, and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and Random loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, June 1997.
4. S. Savari, and E. Telatar, "The behavior of certain stochastic processes arising in window protocols," *preprint*, July 1998.
5. A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," *IEE/ACM Transactions on Networking*, vol. 6, no. 4, August 1998.
6. V. Jacobson, "Congestion avoidance and control," *Proceedings of ACM SIGCOMM '88*, August 1988.
7. J. Bolot, "End-to-end packet delay and loss behavior in the internet," *Proc. ACM SIGCOMM'93*.
8. R. G. Gallager, "Discrete stochastic processes," Boston : Kluwer, 1996.
9. S. Floyd, "TCP and successive fast retransmits," February 1995, URL <ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt>.
10. A. Abouzeid, S. Roy and M. Azizoglu, "Markovian Analysis of TCP/IP Congestion Control for Channels with Random Loss," *submitted to ACM/IEEE Trans. on Networking*, 1999.

Accurate Approximate Analysis of Dual-Band GSM Networks with Multimedia Services and Different User Mobility Patterns*

Michela Meo and Marco Ajmone Marsan

Dipartimento di Elettronica, Politecnico di Torino, Italy
`{michela,ajmone}@polito.it`

Abstract. We describe a technique for the approximate performance analysis of cellular mobile communication networks based on a TDMA scheme (such as GSM) in which the utilization of two separate frequency bands leads to a complex cellular structure with overlapping microcells and macrocells. The analysis technique is based on Markovian assumptions as regards both the traffic flows entering microcells and macrocells and the user mobility pattern, as well as an assumption of flow balance between handovers into and out of any cell.

1 Introduction

The development of new cellular communication networks and services in the countries that adopt the GSM standard is based on the use of two separate frequency bands, around 900 MHz and 1.8 GHz, respectively. Cells served by the 900 MHz band are much larger than cells served by the 1.8 GHz band, due to the much better propagation characteristics of microwaves in the former frequency range through the atmosphere. For this reason, cells served by frequencies in the 900 MHz band are normally called “macrocells”, whereas cells served by frequencies in the 1.8 GHz band are often called “microcells”.

The use of microcells and macrocells in the design of cellular mobile communication networks based on a TDMA scheme yields advantages and drawbacks. The main advantages lie in a much better spatial reuse of frequencies, hence in substantial capacity increases, with the consequent possibility of offering data services at medium-high rates (up to hundreds of kb/s), and even multimedia services through the integration of voice and data traffic flows. The price that has to be paid is mainly a much larger number of handovers during connections. This can be a critical factor, since the design and planning of cellular communication networks aim at meeting specified constraints on handover failure probabilities.

Markovian models have been traditionally used for the design and planning of mobile cellular telephony networks, considering one cell at a time (see for example [1,2,3,4,5]). While this approach proved adequate for networks comprising only macrocells, it cannot be directly transferred to the dual-band environment,

* This work was supported in part by the Italian National Research Council.

where the minimum network element that has to be considered consists of one macrocell and all the microcells comprised within the macrocell. This network element will be called a “cell cluster”. This subsystem is rather complex for the direct development of Markovian models. For this reason, we propose in this paper the adoption of approximate models for the performance analysis of dual-band cellular mobile communication networks comprising microcells and macrocells.

2 System and Modeling Assumptions

A dual-band cellular mobile communication network is considered, where each cell (microcell or macrocell) is served by a different base station. We focus on a particular area, covered by one macrocell and m microcells (one cell cluster), and assume that the macrocell is equipped with $N^{(M)}$ channels, while each microcell is equipped with $N^{(m)}$ channels. Calls taking place in different cells (micro or macro) are assumed not to interfere with each other.

C classes of calls are distinguished on the basis of three aspects: user mobility, call duration, and bandwidth requirements. Bandwidth requirements are expressed in terms of number of channels employed for the connection; let this number be $f(i)$ for class i calls.

A user issues a class i new call request to the base station of the current microcell. If the number of free channels in the microcell is less than $f(i)$, and thus the microcell cannot satisfy the request, the request is forwarded to the macrocell. If the number of free channels in the macrocell is also insufficient for the accommodation of the user request, the request is refused.

In order to simplify the description, we assume that macrocell channels serve only the microcell overflow traffic; however, the model can be easily extended to cope also with requests directed to the macrocell, as is indicated further on.

Users can roam from a microcell (or a macrocell) to neighboring cells while connections are active. Connections are kept alive with handover procedures which try to allocate channels in the new cell according to the same policy adopted for a new call requests. A handover request is preferentially accommodated in the entered microcell, but if the number of free channels is insufficient, the request is forwarded to the macrocell. If even the macrocell cannot serve the handover, the handover request fails.

In the development of the model we introduce the following assumptions:

- The aggregate process of new call requests for class i within a microcell is Poisson, with parameter λ_i (the λ_i 's are estimated from the user population and the system geometry).
- The flow of class i incoming handover requests from other cells is Poisson with rate $\lambda_{h,i}$ (the $\lambda_{h,i}$'s are derived by balancing the incoming and outgoing handover flows, as explained below).
- The class i call duration is an exponentially distributed random variable with parameter μ_i (the μ_i 's are obtained from the user behavior).

- The time between two successive handover requests of a class i call (the call *dwell time*) is assumed to be an exponentially distributed random variable with parameter $\mu_{h,i}^{(M)}$ (in macrocells) or $\mu_{h,i}^{(m)}$ (in microcells) (the $\mu_{h,i}^{(M)}$'s and $\mu_{h,i}^{(m)}$'s are obtained from the user mobility).

3 The Cell Cluster Model

Cells in a cell cluster are studied one by one, taking into account the interaction among adjacent cells by equaling the incoming handover flow and the handover flow out of the considered cell. This approach has been widely and successfully used in the literature when considering individual cells with a single user class, and corresponds to an assumption of independence among the cell behaviors. This independence assumption is justified by the results presented in [6], where it was shown that more complex (multi-cell) models do not lead to significant improvements in the accuracy of performance predictions.

The incoming handover rate of class i calls must be evaluated numerically, since it cannot be a-priori derived from the model parameters. An iterative procedure is used to balance (separately for each user class) the incoming and outgoing handover rates, assuming that the incoming handover rate at step j is equal to the outgoing handover rate computed at step $j - 1$. The iterative procedure is stopped when equilibrium is reached.

Due to the flow balance assumption described above, microcells can be modeled as a set of M/M/K/0 queues with $K = N^{(m)}$ servers representing the channels and C classes of customers representing the calls. For class i calls, the arrival process into each microcell is the superposition of the new call generation process and the incoming handover processes; the resulting total arrival rate is $\lambda_i + \lambda_{h,i}$. The service rate at each microcell is equal to $\mu_i + \mu_{h,i}^{(m)}$.

The macrocell model is slightly more complicated. Users access the macrocell only when not enough channels are available in the microcell where they request service. The traffic that users collectively offer to the macrocell is the superposition of the overflow processes of microcells. The nature of overflow processes and the correlation among queues due to handovers make this traffic non-Poisson. Still, the superposition of a number of overflow processes is expected to reduce the burstiness of the overall arrival process, so that approximating it with a Poisson process may yield acceptable results. In order to study the macrocell in isolation, we thus assume that the superposition of all overflow processes from microcells is Poisson with rate equal to the sum of the microcell overflow rates.

3.1 Microcell Model

The state of a microcell is defined by the vector $\bar{s} = (s_1, s_2, \dots, s_C)$ where each component s_i represents the number of active class i connections.

Assuming that a call of class i requires $f(i)$ channels, the model state space is $\mathcal{S} = \{\bar{s} \mid 0 \leq \sum_{i=1}^C f(i) \cdot s_i \leq N^{(m)}\}$.

The steady-state probabilities $\pi(\bar{s})$ are computed as the product of the single-class solutions (generalized Erlang B formula):

$$\pi(\bar{s}) = \pi(\bar{s}_0) \prod_{i=1}^C \frac{\rho_i^{s_i}}{s_i!} \quad \text{with} \quad \pi(\bar{s}_0) = \pi(0, 0, \dots, 0) = \left(\sum_{\bar{s} \in \mathcal{S}} \prod_{i=1}^C \frac{\rho_i^{s_i}}{s_i!} \right)^{-1} \quad (1)$$

where ρ_i is the class i traffic: $\rho_i = \frac{\lambda_i + \lambda_{h,i}}{\mu_i + \mu_{h,i}^{(m)}}.$

The overflow probability for class i calls, $P_{b,i}^{(m)}$, is given by the sum of the probabilities of states in subset \mathcal{S}_i in which the number of free channels in the microcell is not sufficient to satisfy a class i call:

$$\mathcal{S}_i = \{ \bar{s} \mid \sum_{j=1}^C f(j) \cdot s_j > N^{(m)} - f(i) \} .$$

The rate of class i overflow from a microcell to the macrocell is: $\gamma_i = P_{b,i}^{(m)} \cdot (\lambda_i + \lambda_{h,i}).$ The class i traffic entering the macrocell is given by the superposition of m flows with rate $\gamma_i.$

3.2 Macrocell Model

The macrocell model is similar to the model of a microcell; we just need to use in the macrocell model the appropriate values for traffic parameters.

The main difference between the two models is the input traffic, which for the macrocell is derived from the superposition of the microcell overflow traffics, that were just characterized. The flow of arrival requests at the macrocell is assumed to be Poisson with rate equal to the sum of the overflow rates from microcells. The resulting macrocell traffic of class i is: $\rho_i = \frac{m \cdot \gamma_i}{\mu_i + \mu_{h,i}^{(M)}}.$ If some traffic class directly requests the macrocell resources, the traffic for such class (say class k) becomes: $\rho_k = \frac{\lambda_k}{\mu_k + \mu_{h,k}^{(M)}}.$

The macrocell model state definition is $\bar{s} = (s_1, s_2, \dots, s_C)$, and the macrocell model state space is $\mathcal{S} = \{ \bar{s} \mid 0 \leq \sum_{i=1}^C f(i) \cdot s_i \leq N^{(M)} \}.$

Steady-state probabilities are derived as explained above for microcells [see (1)].

The call blocking probability for class i calls, $P_{b,i}$, is computed as the sum of the probabilities of states in \mathcal{S}_i , in which less than $f(i)$ channels are free in the macrocell:

$$\mathcal{S}_i = \{ \bar{s} \mid \sum_{j=1}^C f(j) \cdot s_j > N^{(M)} - f(i) \} .$$

4 Numerical Results

In this section we discuss some numerical results that were obtained with the proposed modeling approach for a cellular communication network that offers

$C = 3$ classes of service: voice, data, and multimedia services. Voice services are accommodated by connections with just 1 channel. Data services are offered at a higher rate and are thus accommodated by connections with 2 channels. Multimedia services are offered as a combination of a voice and a data connection; hence they are accommodated by connections with 3 channels.

We focus our attention on a cell cluster that comprises $m = 19$ microcells and one macrocell, which are equipped with a number of channels respectively equal to $N^{(m)} = 16$ and $N^{(M)} = 32$.

Performance measures will be plotted versus increasing values of the total input traffic $\lambda = \sum_{i=1}^3 f(i) \cdot \lambda_i$. The input traffics λ_i are assumed to always have values such that the traffic loads of the three classes (i.e. $f(i) \cdot \lambda_i$) are identical. All classes have average call duration equal to 3 minutes (i.e. $1/\mu_i = 3$ min); the average dwell times are respectively $1/\mu_{h,i}^{(m)} = 10/\mu_i$ in the microcell and $1/\mu_{h,i}^{(M)} = 30/\mu_i$ in the macrocell.

In the left part of Fig. 1 we plot the average number of busy channels in each microcell, separating the channels used by each traffic class, but also showing the total value. Observe that, with the chosen values of the request rates λ_i , when the system is not overloaded, all service classes use about the same number of channels. Instead, while approaching overload, the service classes with wider bandwidth requirements suffer because of the increasing difficulty of finding a sufficient number of free channels in the microcell.

A similar behavior can be observed in the right plot of Fig. 1, which shows the channel allocation in the macrocell. The different service classes submit different amounts of traffic to the macrocell; service classes with wider bandwidth requirements experience larger blocking probability at microcells, and thus generate a larger amount of traffic for the macrocell. Under light load conditions the majority of channels in the macrocell is allocated to multimedia calls (class 3 calls). As the traffic increases, the traffics from other classes increase, and the easier access to channels by calls with small bandwidth requirements reverses the distribution of the channel allocation to service classes.

Fig. 2 shows curves of the overflow probabilities at microcells $P_{b,i}^{(m)}$ (left plots), and of the loss probabilities at the macrocell $P_{b,i}^{(M)}$ (right plots). As expected, overflow and loss probabilities are larger for those service classes that require a larger number of channels.

Since channels at the macrocell are the most valuable resource in the cell cluster (they are shared by all microcells as emergency resources in critical conditions), it is worth trying to keep the macrocell as underloaded as possible. For this reason, it may be worth modifying the channel allocation policy at the macrocell so that all connections requests that overflow from microcells to the macrocell are satisfied with just one channel, independently of the service class. This means that services of classes 2 and 3 are temporarily offered with reduced quality. The advantage of such a policy is in the possibility of a significant decrease of the blocking probability for new requests as well as of the handover failure probability. In other words, the call blocking probability and the hando-

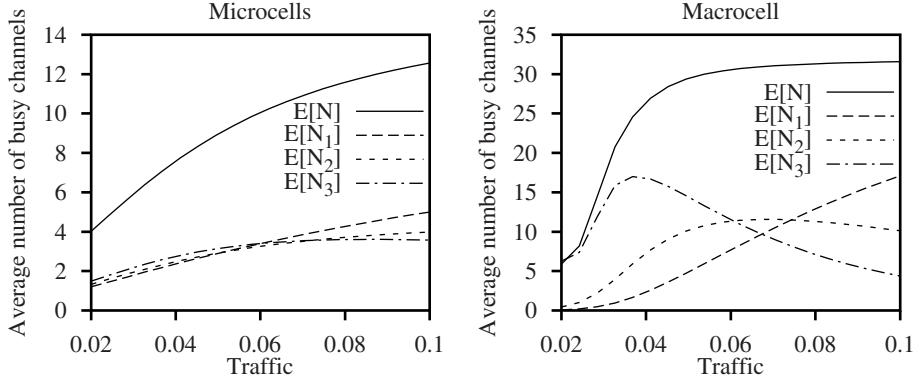


Fig. 1. Average number of busy channels in the microcells and in the macrocell

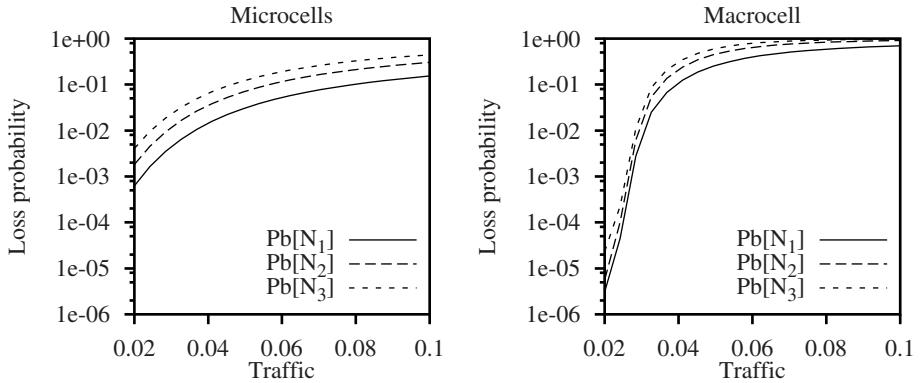


Fig. 2. Call blocking probabilities in the microcells and in the macrocell

ver failure probability decrease at the cost of the temporary deterioration of the QoS for calls with wider bandwidth requirements (class 2 and 3 calls).

Fig. 3 shows the average number of busy channels in the macrocell and the call loss probability that result when the bandwidth restriction in the access to macrocell channels is enforced. It can be observed that the allocation of channels to service classes becomes fair for increasing loads, and that the improvement in the loss probability is remarkable.

5 Conclusions

We proposed a technique for the accurate approximate analysis of cellular mobile communication networks with overlapping microcells and macrocells.

The accuracy of the approximate modeling technique was validated by comparison against detailed simulation results, and found to be quite good. Results were not reported in this paper due to space limitations.

The proposed approximate analysis technique was applied to a network which provides voice, data, and multimedia services by allocating several parallel circuits to the same end user, proving the effectiveness of the approximate analysis

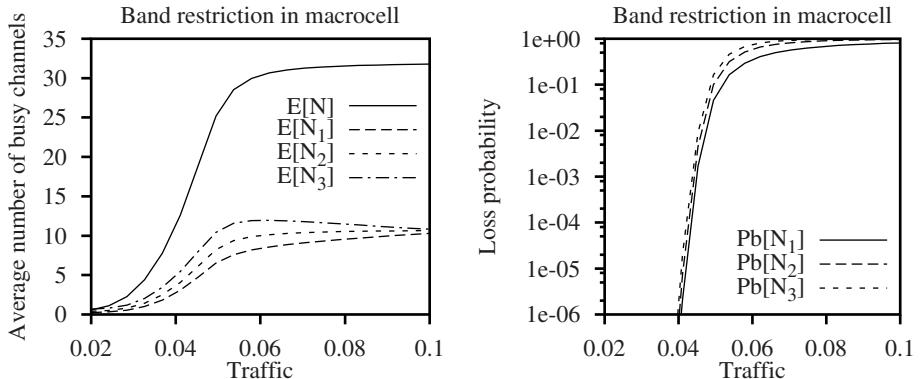


Fig. 3. Average number of busy channels (left plots) and call blocking probabilities (right plots) in the macrocell, with bandwidth restriction

approach in designing cellular mobile communication networks with different resource management algorithms.

The proposed approximate analysis approach can be easily extended to cope with a wide variety of system features that could not be considered in this paper for space limitations; examples are the reservation of channels to handovers, and the repeated attempts by blocked users.

References

1. D. Hong, S. Rappaport, "Traffic Model and Performance Analysis for Cellular Mobile Radio Telephone Systems with Prioritized and Non-Prioritized Handoff Procedures", *IEEE Transactions on Vehicular Technology*, Vol. VT-35, N. 3, pp. 77-92 August 1986.
2. M. Ajmone Marsan, S. Marano, C. Mastroianni, M. Meo, "Performance Analysis of Cellular Communication Networks Supporting Multimedia Services", *IEEE MASCOTS'98*, Montreal, Canada, July 1998.
3. P. Tran-Gia, M. Mandjes, "Modeling of Customer Retrial Phenomenon in Cellular Mobile Networks", *IEEE JSAC Spec. Issue on Personal Communication – Services, Architecture and Performance Issues*, Vol. 15, N. 8, pp. 1406-1414, October 1997.
4. K. K. Leung, W. A. Massey, W. Whitt, "Traffic Models for Wireless Communication Networks", *IEEE JSAC*, Vol. 12, N. 8, pp. 1353-1364, October 1994.
5. Y. Lin, "Modeling Techniques for Large-Scale PCS Networks", *IEEE Communication Magazine*, Vol. 35, N. 2, pp. 102-107, February 1997.
6. M. Ajmone Marsan, G. De Carolis, E. Leonardi, R. Lo Cigno, M. Meo "How Many Cells Should Be Considered to Accurately Predict the Performance of Cellular Networks?", *European Wireless'99*, Munich, Germany, 6-8 October 1999.

Paging Strategies for Future Personal Communication Services Network

Partha Sarathi Bhattacharjee¹, Debasish Saha² and Amitava Mukherjee^{3*}

¹Department of Telecommunications,
Telephone Bhawan 34 B.B.D. Bag, Calcutta 700 001, INDIA
Tel: 91-33-2207070 and Fax: 91-33-2207070

²Department of Computer Science & Engg.,
Jadavpur University, Calcutta 700 032, INDIA.
d.saha@computer.org

³PricewaterhouseCoopers Ltd
Plot Y14, Block EP, Sector 5 Salt Lake, Calcutta 700 091, INDIA
Tel & Fax: 91-33-357-3384& 91-33-357-3394
amitava_mukherjee@in.pwcglobal.com
(* Author for correspondence)

Abstract: This paper presents two intelligent paging strategies for a Personal Communication Services Network (PCSN). They are termed as sequential intelligent paging (SIP) and parallel-o-sequential intelligent paging (PSIP). Unlike the conventional blanket paging where all cells in a location area are polled at a time, in SIP (PSIP) one cell (a group of cells) is polled at a time. The cells to be polled are determined from the occupancy probability vector. The objective is to reduce the signaling load. The proposed schemes lead to a decrease in paging signaling load at the cost of some extra processing power. When high intensity traffic is expected, PSIP is always preferred to other paging schemes. However, when incoming traffic rate is low, SIP performs better when paging cost per cycle is the criterion for choosing a particular scheme of paging. The efficacy of these two intelligent paging strategies is shown with the help of simulation results.

I. INTRODUCTION

Personal Communication Services (PCS) [1]-[4] support personal mobility based on personal number, terminal mobility provided by wireless access and service portability through management of user service profiles. Thus, PCS will lead to ubiquitous availability of services to facilitate the exchange of information between nomadic end users, independent of time, location and access methods. We consider a hierarchical structure of PCS network (PCSN) [3] where the total service area (SA) is divided into a number of location areas (LAs). Each LA is further subdivided into a number of cells. For each cell, there is a base station (BS). The function of BS is to provide the radio link to the MTs within the cell corresponding to the BS. The BSs within an LA talk with each other through a mobile switching center (MSC). With the advent of third generation mobile telecommunication systems, Universal Personal Telecommunication (UPT) [5], Universal Telecommunication systems (UMTs) [6] are now in the offing which enable each UPT user to participate in a user-defined set of subscribed services and to originate and receive calls on the basis of unique personal, network independent UPT number across multiple networks at any terminal, independent geographic location.

The objective of a successful paging is to establish a wireless signaling association between the MT and the network. In Second Generation Mobile Communication Systems (SGMCTs) the issue of finding an MT is treated as follows. The network keeps track of the location of every attached MT with the accuracy of an LA. A location update in the database takes place whenever an attached MT crosses the boundaries of an LA. Whenever an incoming call arrives the network has to locate the called MT within the precision of a cell i.e. to determine the base station via which a wireless signaling link between the MT and the network can be established. During paging, a specific message is broadcast simultaneously via all BSs over the whole LA so as to make alert the called MT i.e., paging area is equal to the LA. The MT, upon receiving the paging request responds to the BS with the stronger received signal strength. Then a wireless link between the called MT and the network is established. This completes process of locating an MT. Since paging amounts to issuing queries about location of the called MTs, these queries require signaling messages. Since the boundaries of the LAs are fixed, MTs moving with high velocities will register more frequently or require larger LAs, which entail higher paging cost. It is also evident that the paging signaling overhead is proportional to the size of the location area in conventional polling [9]. With the increase in number of service and number of MTs in service, the radio spectrum will become a scarce commodity. This calls for a reduction in signaling load between MTs and BSs in order to make more bandwidth available for voice, video and data traffic. So a more efficient paging strategy is now necessary. One of the key issues addressed here is to deploy the methods of intelligent paging, which results in consequent reduction in signaling load associated with paging.

The growing demand for PCS and finite available bandwidth motivated several investigations into the methods of delivering calls. A scheme called reverse virtual call setup (RVC) which requires a few new networks SS7 signaling

messages was proposed in [10]. RVC can function within the existing cellular paging network or with an integrated overlaid paging network. A method that saves paging signaling load by exploiting information related to the MT location during the most recent interaction between the MT and the PCSN was suggested in [11]. The delay time for paging and paging response time were analyzed in [12]. A selective paging scheme for PCSN was proposed by Akyldiz et al. [13] Which modeled the movement of MTs as one-dimensional and two dimensional hexagonal, mesh random walk. A state based paging technique has been dealt in [14]. While variation of optimum total cost with call-to-mobility ratio has been discussed in [13], LA planning based on time zones and categories of MTs is presented in [11]. Average paging/registration cost rate incurred in greedy registration procedure is compared with a timer-based method in [14]. Methods of reduction of paging signaling load were not discussed in [10]-[14]. Our paper addresses this issue.

Section I deals with the review of previous works, motivation behind the work and our contribution. Section II describes the methodology. Section III discusses the sequential intelligent paging (SIP) and section IV deals with parallel-o-sequential intelligent paging. Simulation results and related discussions have been presented in section V. Section VI sums up the entire work.

II. METHODOLOGY

The following notations are used in this paper.

NOTATIONS:

S = Total number of cells in an LA

v_{\max} = Maximum speed of a mobile terminal (kmhr^{-1})

v_{\min} = Minimum speed of a mobile terminal (kmhr^{-1})

\bar{v} = Average speed of MT (kmhr^{-1})

ρ = Density of MTs (km^{-2})

μ = Average number of incoming calls per MT per hour

λ_{pg} = Paging rate

ϕ = call-to-mobility ratio (CMR)

A_{cell} = Area of a cell

K^{conv} = granularity factor in conventional paging

K^{SIP} = Granularity factor in sequential paging

K^{PSIP} = Granularity factor in parallel-o-sequential intelligent paging

S^{conv} = Number of times an MT is paged in conventional paging before it is found

S^{SIP} = Number of times an MT is paged in sequential paging before it is found

S^{PSIP} = Number of times an MT is paged in parallel-o-sequential intelligent paging before it is found

τ^{conv} = Paging delay in conventional paging

τ^{SIP} = Paging delay in sequential paging

τ^{PSIP} = Paging delay in sequential-o-parallel intelligent paging

T_pB_p = Time bandwidth product for paging messages

C_p^{conv} = Paging cost in conventional paging

C_p^{SIP} = Paging cost in sequential paging

C_p^{PSIP} = Paging cost in parallel-o-sequential intelligent paging

P = Occupancy probability vector

p_i = Probability of finding the MT in the i th cell

P_{SFP} = Probability of successful first paging

The movement of MTs is modeled according to some ergodic, stochastic process. In order to quantitatively evaluate, the average cost of paging, time varying probability distributions on MTs are required. These distributions may be derived from the specific motion models, approximated via empirical data or even provided by the MTs in the form of partial itinerary at the time of last contact. In purely sequential polling one cell is polled at a time. Sometimes, instead of polling one cell at a time, we go for polling a cluster of cells in an LA, called parallel-o-sequential intelligent polling (PSIP) which is a special case of sequential intelligent polling (SIP). At the instance of a call meant for to be terminated to an MT, which roams within a certain LA, paging is initially performed within a portion of LA, which is a subset of the actual LA. This portion of the LA, which is a set of base stations of paging (BSPs), is called a paging area (PA). Intelligent paging is a multi-step paging strategy which aims at determining the proper PA within which the called MT currently roams.

We now define granularity factor (K), which shows fineness in polling. In general, we define granularity factor as

$$K=(\text{number of cells to be polled})/(\text{number of cells in an LA})$$

The maximum value of granularity factor is 1 i.e., when all cells in an LA are polled in one polling cycle. Hence, $K^{SIP}=1/(number\ of\ cells\ in\ an\ LA)$. As the allowable paging delay is the constraint governing the number of paging steps, sometimes we go for a PSIP the cells where a cluster of cells are polled at a time i.e., granularity factor is K^{PSIP} where $1>K^{PSIP}>K^{SIP}$. The PSIP aims at reducing the paging load within an LA without reducing the LA size and keeping the delay under tolerable limits.

In city area [11], while considering the mobility pattern of an MT, the drift i.e. the change in direction and the change in speed both are to be considered simultaneously to determine the possible location of the called MT. Based on biased sampling, the pdfs of speed and direction of cell-boundary crossing MTs are furnished in [16]. Depending on the drift and the speed, the cells in inner circle then next outer circle are polled till the MT is found. For highly mobile MTs and large LA, the number of cells to be polled increases significantly. We specifically assume that

- i) probability density function on location of MTs is known and considered to truncated Gaussian pdf
- ii) Time elapsed since the last known location.
- iii) The paging process described here is rapid enough to the rate of motion of MT i.e., MT to be found, does not change its location during the paging process.
- iv) Call arrival process has a Poission distribution
- v) Call holding time is an exponential distribution
- vi) MTs are allowed to alter directions only at crossroads. The same direction is kept with probability, 0.5, while MT turns to left/right with an equal probability, 0.25.

We apply the street unit model [11] where the MTs are assumed to be moving according to some ergodic, stochastic process. The street network has two types of streets namely, i) multi lane highway ii)cross roads with uni-directional vehicular flow. In one dimensional version of Brownian motion, an MT moves by one step Δx to the right with some probability, p and to the left with probability q, and stays there with probability (1-p-q) for each time step Δt . Given the MT starts at time $t=0$ for position $x=0$, the Gaussian pdf on the location of an MT is

$$P_{X(t)}(x(t)) = (\pi D t)^{0.5} e^{-\frac{(x-vt)^2}{4Dt}}$$

Where $v=(p-q)*(\Delta x/\Delta t)$ is the drift velocity and $D=2((1-p)p+(1-q)q+2pq)(\Delta x)^2/\Delta t$ is the diffusion constant, both functions of the relative values of time and space steps. We then derive the occupancy probability vector of current location of MTs. The probability of occupancy will be obtained by integrating the density function over the segment associated with the location of an MT. The probability of occupancy of MT in different cells is determined. The cell(s) to be polled depends on the probability occupancy vector and the allowable delay.

III. SIP

The SIP strategy described here, aims at the significant reduction in load of paging signaling on the radio link. In SIP, one cell is polled at a time and the process continues till such time the called MT is found or timeout occurs whichever is earlier. The selection of the cell to be polled sequentially depends on the determination of occupancy probability vector, which is based on the stochastic modeling delineating the movement of the MT. When the paging is unsuccessful during a polling cycle the MT is paged in other cells of the LA sequentially which have not been polled so far. This phase is completed in one or more than one paging step(s).

The paging rate represents the average number of paging packets, which arrives at a base station during unit time. In conventional or blanket paging, upon arrival of an incoming call, all cells are polled at a time for locating the called MT i.e., each MT is paged S times before the called MT is discovered. Hence, paging rate in a cell becomes $\lambda_{pg} = \mu p S A_{cell}$.

In SIP scheme, each paging request is sent to those BS where there is maximum probability of finding the called MT. A forward signaling channel is a common signaling channel assigned to any multiplexed stream of paging and channel allocation packets for BS to MTs. A typical value of an FSC slot is 18.46 ms. Paging rate in a cell per hour becomes a staggeringly high figure of the order of 10^5 for $p \sim 100 \text{ km}^{-2}$, $A_{cell} \sim 1 \text{ km}^2$, $S \sim 20$ and $\mu \sim 10 \text{ calls hour}^{-1}$. Following GSM approach, it will be very difficult to accommodate the paging requests in FSCs unless number of paging packets are increased which will lead to a consequent decrease in number of channel allocation packets and thus results in an increase in call blocking probability. Hence in SIP, the PRs are stored in a buffer in MSC and depending on the occupancy probability vector, a particular BS receives the PR for a particular called MT. The paging cost per polling cycle in this scheme is $C_p^{seq} = S A_{cell} \mu p T_p B_p$.

The sequential paging algorithm is given below.

STEP 1: When an incoming call arrives, calculate the occupancy probability vector, [P] based on the probability density function, which characterizes the motion of the MT.

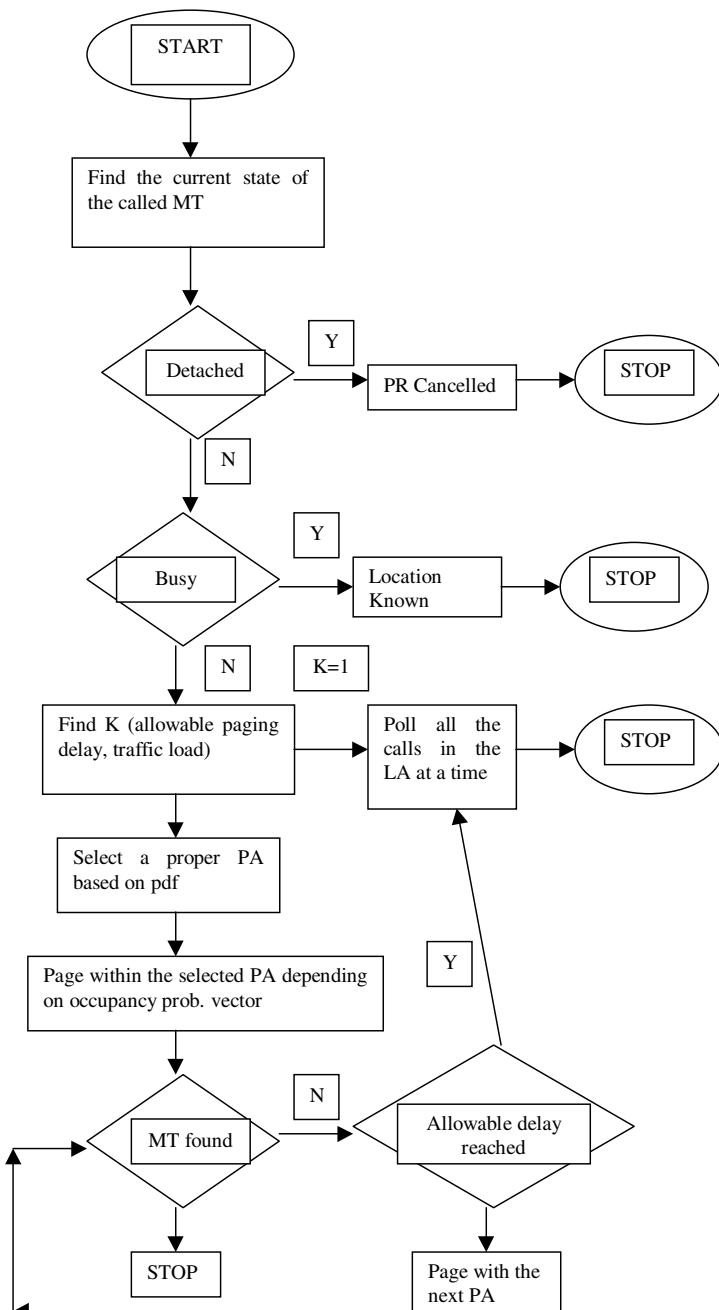
STEP 2: Sort the elements of [P] in descending order.

STEP 3.0: FLAG = False;

STEP 3.1: Poll the ith cell for which $p > p_j$ for $i, j \in S$
 STEP 3.2: if the MT is found
 FLAG = True;
 Go to ENDSTEP ;
 STEP 4.0:
 If time-out occurs
 Go to ENDSTEP;
 Else
 Poll the jth cell where $j=1,2,3\dots$ & $j \neq i$
 Go to STEP 3.2;
 Endif
 ENDSTEP: If FLAG = True
 Declare : "Polling is Successful" ;
 Else
 Declare " Polling is Unsuccessful" ;
 Endif
 STOP.

IV. PSIP

The benefit that accrues out of PSIP is the overwhelming reduction in paging cost and signaling load. From the performance point of view, the quality of service in terms of paging delay should be similar to the GSM like approach and the processing overhead to exploit the paging related information should not be considerable. Although, the method proposed here, requires some additional storage, it does not impose any additional transaction to the network database since the retrieval of paging related information can be performed in conjunction with the retrieval of user profile during the call setup phase. So PSIP allows for defining larger LA. This in turn leads to a reduction in number of location updates. The number of steps in which the paging process should be completed i.e., the MT is to be found, depends on the allowed delay during paging. In the very first phase, the network decides whether the appropriate type of paging i.e. blanket paging (GSM like approach) or multiple step paging. The network then examines whether paging is needed by checking the current status of the MT. An MT can be switched off so as to make it unreachable. This means not only the MT does not want to make or receive any call, but also the network itself cannot detect the current position of the MT. An MT, which has been switched off, may move into a new LA or even into another network operator's area. When switched on again, the MT should inform the network about its status and location. This procedure is called attachment. If it is detached, the paging request (PR) is cancelled. If it is busy, a relation between the MT and the network already exists and therefore paging is not required. If it is free, the network proceeds for paging upon receipt of a PR. The network examines whether the multiple steps paging strategy should be applied or not. The decision is based on the allowable paging delay and the current traffic load which when exceed a threshold value, a multiple step intelligent paging is applied. The flowchart of PSIP is in Figure 1. Continuous unsuccessful paging attempts may lead to unacceptable network performance in terms of paging delay. To minimize the number of paging steps, the network should guarantee that the P_{SFP} is high (typical value for e.g. 90%). The PA should consist those cells where sum of probabilities of finding the called MT is greater than or equal to the typical value chosen for P_{SFP} . The paging cost per polling cycle in PSIP is $C_p^{\text{PSIP}} = K^{\text{PSIP}} S A_{\text{cell}} \rho \mu T_p B_p$.

Figure 1: Flow chart for PSIP

V. RESULTS

Simulation results have shown that SIP and PSIP achieve paging signaling load reduction of the order of 60% or higher compared to the blanket polling applied in GSM. The results presented in this section are based on a two- lane highway and crossroads of a street unit model. The length of each cell is 5km. And the width is 0.2km. The vehicular traffic moving along the two lane- highway have maximum speeds 70kmhr^{-1} , 20 kmhr^{-1} and minimum speeds 30kmhr^{-1} , 10 kmhr^{-1} respectively. The occupancy probability vectors for $\mu=3\text{ calls hour}^{-1}$ are furnished in Table I.

Table I: Probability occupancy vectors in a two lane highway

$v_{\max}=70\text{km hour}^{-1}$, $v_{\min}=30\text{km hour}^{-1}$, average speed = 50 km hour^{-1} , $\mu=3\text{ calls hour}^{-1}$

Cell no.	1	2	3	4	5	6	7	8	9	10
P(Lane-1)	17.8×10^{-4}	4.57×10^{-2}	2.93×10^{-1}	4.56×10^{-1}	1.84×10^{-1}	1.83×10^{-2}	4.6×10^{-4}	2.2×10^{-5}	1.5×10^{-5}	2.1×10^{-6}
P(Lane-2)	0.3950 7	0.49897	0.00187	1.8×10^{-4}	1.2×10^{-4}	1×10^{-4}	9.1×10^{-4}	8.5×10^{-4}	7.2×10^{-4}	6.1×10^{-4}

Table II : Conventional paging versus SIP for lane-1

Number of cells per LA	10
Density of MTs	20
CMR	0.06
K^{SIP}	0.1
Paging channel per base station	8
S^{conv}	10
S^{sip}	3
Percentage decrease in signaling	70

Table III: Signaling load vs. CMR in sequential paging

Call-to-mobility ratio	% decrease in signaling load in sequential paging
0.02	60
0.06	70
0.08	70
0.16	80
0.32	90

Table II shows a comparison between blanket paging and sequential paging for CMR=0.06 for lane -1. In blanket paging all 10 cells in the LA are to be polled before a called MT can be located whereas in sequential paging the cells with greater probability of occupancy are polled until $P_{\text{SFP}}=0.9$. Thus signaling load is reduced by 70%. The decrease in signaling load in SIP over conventional paging is evident from Table III. For CMR =0.32, 90% reduction in the signaling load during paging can be achieved by this scheme.

Table IV: Probability occupancy vector in a one-way crossroad $v_{\max}=30\text{km hour}^{-1}$, $v_{\min}=10\text{km hour}^{-1}$, average speed = 20 km hour^{-1} , $\mu=1\text{ call hour}^{-1}$ at the crossing , $\Pr[\text{left}]=\Pr[\text{right}]=0.5 \times \Pr[\text{straight motion}]$

Cell no.	1	2	3	4	5	6	7	8	9	10
P	5.37×10^{-2}	1.12×10^{-1}	1.829×10^{-1}	2.338×10^{-1}	1.169×10^{-1}	9.145×10^{-2}	5.845×10^{-2}	4.57×10^{-2}	5.84×10^{-2}	4.57×10^{-2}

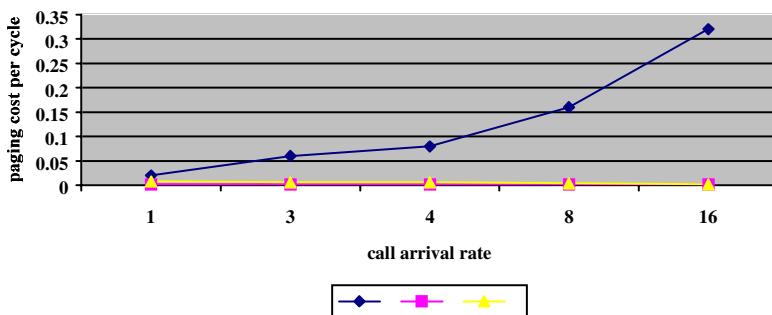
Table V: Comparisons of paging cost per polling cycle

μ	τ^{conv}	C_p^{conv}	τ^{seq}	C_p^{SIP}	τ^{PSIP}	C_p^{PSIP}
1	1	0.02	4	0.002	1	0.008
3	1	0.06	3	0.002	1	0.006
4	1	0.08	3	0.002	1	0.006
8	1	0.16	2	0.002	1	0.004
16	1	0.32	1	0.002	1	0.002

Table V shows a comparison of delay and paging cost in conventional paging, SIP and PSIP. When rate of incoming call is less, it is observed that the paging delay in sequential paging is more and three or four polling cycles are required before the called MT is found. In PSIP, the P_{SFP} is ensured to be 0.9 by polling the required number of select cells and the called MT is located. A comparison of paging costs in conventional paging, SIP, PSIP is shown in Figure 3. With the increase in CMR, the cost of polling increases proportionately in conventional paging whereas it is always minimum in SIP. In PSIP, for low traffic it is more than that of SIP. But it approaches the cost in SIP as CMR increases.

VI. CONCLUSION

Two intelligent paging strategies namely SIP and PSIP have been presented in this paper. The reduction in paging signaling load with the increase in CMR is highlighted also. The paging cost per polling cycle, delay associated with the process of SIP and PSIP, have been studied for low and high intensity traffic conditions. We conclude that in scenarios where high intensity traffic is expected PSIP is always preferred to other paging schemes. However, when incoming traffic rate is low, SIP performs better when paging cost per cycle is the criterion for choosing a particular scheme of paging. As the paging cost increases monotonically with each unsuccessful polling cycle, the better option is to adopt PSIP. The efficacy of PSIP strategy as far as paging signaling load is concerned, is directly related to the capability of the network to accurately predict the location of the called MT. As both the schemes presented here achieve a significant reduction of the paging signaling load compared to the technique applied in GSM there is room for defining larger LAs which will lead to minimization of location updating signaling load on the network. Thus, the paging methods presented here are useful for future PCSN.

**Fig. 2: Variation of paging cost per cycle with CMR**

References:

- [1] R.H.Katz, "Adaptation and mobility in wireless information system", IEEE Personal Commn., vol.1, no. 1, pp6-17, 1994.
- [2] D.C. Cox, "Personal communications - viewpoint", IEEE Commun., Nov 1990.
- [3] B. Jabbari, "Intelligent network concepts in mobile communications", IEEE Commun., pp 64-69, Feb 1992.
- [4] J Sarnecki et al. "Microcell Design Principle" IEEE Comm. Magazine vol:34 ,no.:4, April'93.
- [5] J.G.Markoulidakis, "Mobility Modeling in Third Generation Mobile Telecommunication Systems", IEEE PCS, August'97.
- [6] J.G.Markoulidakis, G.L.Lyberopoulos, D.F.Tsirkas, E.D.Skyas, "Evaluation in LA Planning in Future Mobile Telecommunication Systems", Wireless Network 1995.
- [7] J.A.Audestad, "GSM General Overview of Network unctions", in Proc. Int. Conference Digital Land Mobile Radio Communication, Venice, Italy, 1987.
- [8] A.Modarressi, R.Skoog, "Signaling System no. 7: A Tutorial", IEEE Comm. Magazine, vol.28, no.7, pp9-35, July, 1996.
- [9] P.S.Bhattacharjee, D.Saha, A.Mukherjee, "Determination of optimum size of a location area" , COMM_SPHERE'99 ,France.
- [10] Chih-Lin, G.P.Pollini, R.D.Gitlin, "PCS Mobility Management Using the reverse Virtual Call Setup Algorithm, IEEE/ACM Trans. on Networking, vol. 5, no. 1, February'97
- [11] J.G.Markoulidakis et al., "Mobility Modeling in Third Generation Mobile telecommunication Systems", IEEE,PCS, August'97 .
- [12] Izhak Rabin, Cheon Won Choi, " Impact of Location Area Structure on the Performance of signaling Channel in Wireless Cellular Networks, IEEE Comm Magazine, February '97.
- [13] I.F.Akyildiz, J.S.M. Ho, Y.B.Lin, "Movement Based Location Update and Selective Paging for PCS Network", IEEE/ACM Trans on Networking, vol.4, no.:4, August'96.
- [14] C.Rose, " State Based Paging/ Registration: A Greedy Technique", Win-lab TR 92Rutgers Univ., Dec., 1994.
- [15] A.Papoulis, "Probability, random Variable and Stochastic Processes", Mc. Graw Hill, 3rd. edition, New York.
- [16] M.M.Zonoozi, P.Dassanayake, "User Mobility and Characterization of Mobility Patterns", IEEE on Selected areas in Communication, vol.15, no.7, September'97.

A Framework for Matching Applications with Parallel Machines

J. In, C. Jin, J. Peir, S. Ranka, and S. Sahni

Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611
`{juin, chjin, peir, ranka, sahni}@cise.ufl.edu`

Abstract. This paper presents a practical methodology for performance estimation of parallel applications on various supercomputers. Instead of measuring the execution time of the applications with different number of processors on each machine, we estimate the time based on characterization of the workloads. Benchmarking computation and communication primitives in the applications is also performed on parallel computer systems for the characterization. Finally, a performance model is constructed for performance estimation, and verified with different number of processors on each target system.

Our results show that accurate performance estimation is possible with the model constructed using the workload characterization method. With the result figures, we discuss the reasons why over or under estimations occur on each target machine.

1 Introduction

A typical high-performance computing environment normally consists of a suite of high-performance parallel machines and workstations connected by a high-speed interconnection network. Efficient scheduling and utilizing the available computing power are essential to maximize the overall performance in such an environment. Good performance for a given application on a parallel machine requires a good mapping of the problem onto the machine. Getting this mapping implies choosing an appropriate machine based on the application requirements. Further, many applications require integrating algorithms from diverse areas such as image processing, numerical analysis, graph theory, artificial intelligence, and databases. These problems may not be able to solve efficiently on one parallel machine because they consist of several parts, each of which requires differing types and amounts of parallelization. For such applications it may be required that different parts of the application code are executed on different machines available in a given computing environment.

There is a need for the development of an expert system tool to assist the user in effective matching of his/her application to the appropriate parallel computer(s). Ideally, we would like to design and develop a performance modeling and prediction tool that will allow users to obtain code fragments (if any) with

performance or scalability bottlenecks, and derive performance for different target machines for a range of number of processors. Such a tool will allow users to develop scalable code without actually executing codes on parallel machines to derive performance.

Our goal is to be able to model the performance of a given software on a variety of different architectures. Towards the above goal, we targeted the performance assessment of a set of parallel applications on various supercomputers. Instead of measuring the run time of the applications on each machine with different number of processors, we estimated the execution time based on workload characterization. Performance measurement was used only for validation of the modeling and prediction results.

The literature abounds with work on benchmarking computation and communication primitives and estimating the performance of parallel algorithms on supercomputers. An accurate static performance estimation of parallel algorithms is possible by using basic machine properties connected with computation, vectorization, communication and synchronization [3]. Trace-driven [6, 1] and execution-driven [2] simulations are also popular for studying detailed processor performance on both uniprocessor and multiprocessor environment. Particularly, this simulation methodology is used to investigate how to improve the performance of the shared-bus, shared-memory systems [4]. In order to avoid complexities with trace-driven simulation, workload characterization techniques are used which can derive behavior of systems like inter-clustered structure, and finally estimate the performance without detailed trace simulations [8].

The rest of the paper is as follows. In Section 2, we describe the modeling of communication overhead. Section 3 describes the workload characterization. Section 4 presents experimental results. We conclude in Section 5.

2 Benchmarking and Characterization of Workload

Under the SPMD model of computation, a parallel program is partitioned into a set of *computation* and *communication* regions. A computation region can be defined as a program segment that is separated by proper synchronization and data communication primitives, while the communication region contains data communication instructions to send and receive data among different processors to satisfy the data dependence. There are several steps involved in modeling and projecting performance of parallel applications on parallel systems (Fig. 1). In order to estimate the execution time of a program on different parallel systems, we first need to benchmark the program on a base parallel system, and to characterize the workload that includes the amount of computation required for each processor in each computation region as well as the data communication requirements throughout the execution of the program. The second step is to build the performance model that consists of extrapolation of the execution time of the computation regions on the target parallel systems and estimation of the communication times based on the empirical communication model built for each target system. Thirdly, the performance models are verified by com-

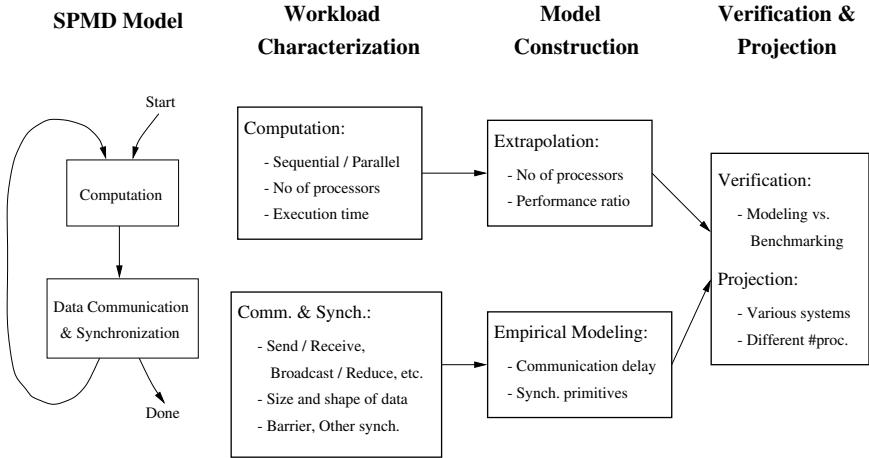


Fig. 1. Method of workload characterization and performance projection

paring the estimated execution time with the real measured time on the target systems. Finally, we can use the model to project the performance of the selected application on different parallel systems with a range of number of processors.

2.1 Example Application

We use a finite-difference program for modeling and projection.¹ In order to estimate the execution time of this program on a CRAY T3E, IBM-SP and ORIGIN 2000, we first benchmark and characterize the workload on an 8 processor Sun/Enterprise system.

Three systems, IBM-SP2, CRAY-T3E and SGI-Origin2000, are used as target systems in this study. The IBM-SP2 system consists of 256 135 MHZ IBM RS/6000 POWER2 Super Chip (P2SC) processors, and peak computational speed of each processor is 540 MFLOPS. The system has 256 nodes, 256 gigabytes(GB) of memory in total and 2 high-Performance Parallel Interfaces(HiPPI) [5]. The CRAY T3E has 544 Processing Elements(PEs), and each PE includes a 600-MHZ DEC Alpha 21164 CPU and 256 MB of memory. The system at CEWES has 300 GFlops peak performance [9]. The SGI Origin 2000 supports up to 512 nodes which can be interconnected by a scalable network. Each node has one or two R10000 processors. A node contains up to 4GB of coherent memory, its directory memory and a connection to IO subsystem [7].

The selected program uses MPI primitives to implement parallel computation. The program divides the computational grid across the processors. The number of processors is provided as an input argument. Each processor computes its portion of the grid, and communicates with other processors to send and receive necessary results. At the end of each time step, processor zero collects

¹ This code was provided to us By Dr. Fred Tracy at CEWES/MSRC. The main computational routine takes above 98% of the total execution time. This was the routine used in our modeling effort.

the results from other processors. The program can be partitioned into three parts: the initialization phase, the main computation phase, and the output and house-keeping phase. The MPI-based time routine, MPI_WTIME(), is inserted between phases to measure their execution time.

The main computation routines are partitioned into 19 computation regions and 4 communication regions, and each region is primarily separated by barrier synchronizations (MPI_Barrier). For modeling, each region may be further partitioned into sub-regions depending on the structure of the program. In the communication region, four MPI primitives: broadcast (MPI_Bcast), reduce (MPI_Reduce), send (MPI_Send), and receive (MPI_Recv) are used for data communication among processors.

2.2 Modeling Communication

In our study, several MPI primitives are benchmarked and modeled. The selected application includes the following four major primitives; MPI_Send, MPI_Recv, MPI_Bcast, and MPI_Reduce. Benchmarking the primitives is considered with 2, 4, 8, 16, 32, or 64 processors respectively, and input data type is fixed with floating point type. We use our benchmarking results to derive formulas for empirical modeling in communication. In the modeling, we first derive a formula for each given number of processors, then derive a simplified formula for an arbitrary number of processors.

2.3 Modeling Computation

Loop iteration is the basic structure used for modeling. We categorize loops into two types, simple and complex, based on the difficulty of modeling. Simple loops require execution time proportional to the total number of iterations. All other loops are considered to be complex.

Simple Loops There are two main scenarios for modeling simple loops on multiprocessors. In the first case, the number of iterations on each processor is inversely proportional to the number of processors. Given the measured execution time on a uniprocessor environment, we can model the execution on multiple processors based on the following simple formula.

$$M = E \times \lceil I/P \rceil . \quad (1)$$

where M is the execution time with multiprocessors, E is the execution time of one loop iteration on one processor, I is the number of iterations with one processor, and P is the number of processors. Note that when the total number of iterations cannot be evenly divided by the number of processors, the smallest integer which is greater than the (I/P) is used.

In the second case, each processor executes the same number of iterations as the computation is replicated on all the processors.

Complex Loops In the selected finite-difference program, a complex loop structure is encountered in several subroutines. In some cases, the loop iteration goes up with the number of processors. For a more accurate modeling, we insert counters into the complex loops. It helps measure the precise iteration numbers for different number of processors.

Avoiding timers to measure the execution time of small computation regions inside an iteration loop, we actually measure the time for the entire loop. We can then model the execution time by solving a set of equations based on the number of iterations of these small regions.

The following equations show an example. Suppose the execution times of an iteration loop are A, B, C and D measured on four different processors, and the execution times of small loops and conditional statements in the loops are T_ngh, T_iroot, T_true, and T_false when the loop iterates one time. Also we assume the numbers of iterations for the small regions are equal to Xs, Ys, Zs, and Ws. Then we can solve the equations to obtain T_ngh, T_iroot, T_true, and T_false.

$$A = X1 * T_{ngh} + Y1 * T_{iroot} + Z1 * T_{true} + W1 * T_{false} . \quad (2)$$

$$B = X2 * T_{ngh} + Y2 * T_{iroot} + Z2 * T_{true} + W2 * T_{false} . \quad (3)$$

$$C = X3 * T_{ngh} + Y3 * T_{iroot} + Z3 * T_{true} + W3 * T_{false} . \quad (4)$$

$$D = X4 * T_{ngh} + Y4 * T_{iroot} + Z4 * T_{true} + W4 * T_{false} . \quad (5)$$

Note that the actual number of equations can be very large due to the fact we can measure the execution time of the outside loop on different processors with various number of total used processors. In order to increase the estimate accuracy, we use the statistical analysis tool, SAS to obtain more accurate timing for these small regions.

Compiler optimizations The three systems, CRAY T3E IBM-SP2 and SGI Origin 2000 have several levels of compiler optimization respectively, and the default level of each is different. When the performance of a program is compared on the systems, the compiler optimization level is required to be same.

In the selected program, A few complex loops are restructured by the compiler optimization. As a result, the total number of the execution of the loops may be much smaller than the number which the high-level code indicates. This is especially true with a large number of processors.

The situation is further accentuated by the fact that the number of iteration in a loop in a ‘measured’ execution with inserted iteration counters does not match with the actual number in the ‘real’ execution. This is due to the fact that when counters are inserted, the compiler can no longer restructure the loop. For the purpose of this study, instead of rewriting the loop code, we insert dummy counters inside the loops to prevent the compiler from restructuring the loop.

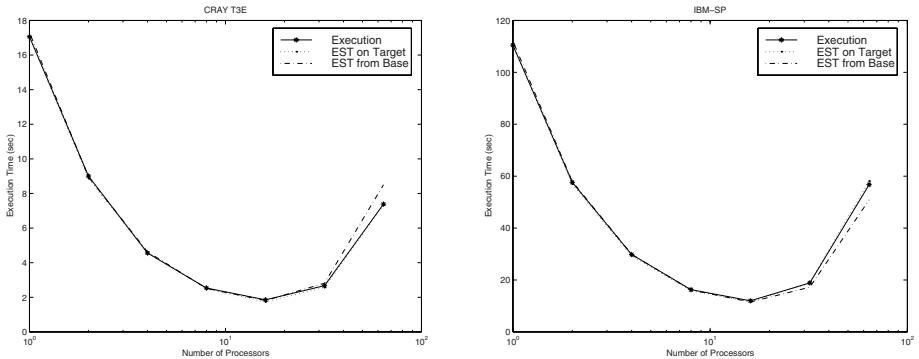
Table 1 shows the differences in execution time with and without dummy counters inserted in a loop statement of a selected application. We can observe

Table 1. The execution time with/without dummy counter (optimization: default)

No. of Procs	Cray T3E		IBM-SP		Origin 2000	
	W/O (s)	With (s)	W/O (s)	With (s)	W/O (s)	With (s)
1	17.04	17.06	110.5	110.6	108.3	106.2
2	9.002	9.008	57.52	57.76	55.93	54.86
4	4.544	4.559	29.77	29.92	28.63	28.09
8	2.459	2.534	16.21	16.45	15.20	15.37
16	1.516	1.854	11.98	12.01	10.01	10.09
32	1.182	2.672	16.77	18.86	12.77	14.39
64	3.078	7.382	50.87	56.74	34.20	38.95

that the difference becomes bigger as the number of processors grows. The values in table 1 result from compiling the application program with default optimization level.

3 Model Verification and Performance Projection

**Fig. 2.** Performance estimation on CRAY-T3E and IBM-SP2 (EST: Estimation)

We first develop and verify our performance model on a local SUN/Enterprise system with 8 processors. This is used as the base machine for modeling and extrapolating the time to other machines. We show the results of two approaches for the targeted machines. First, we estimate the execution times of a subroutine based on the performance model built on the base system. In order to extrapolate the execution times on the target systems, we used a *performance ratio* between the base and the target systems by comparing the execution time of the selected application running on a uniprocessor environment. Second, we also estimate the performance of the subroutine based on the model built on each target system.

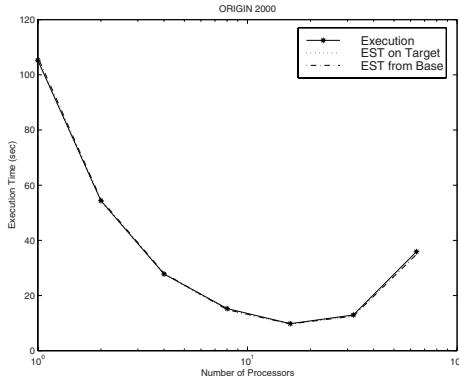


Fig. 3. Performance estimation on SGI-Origin2000 (EST: Estimation)

Basically, it is the same as constructing the model on the base and the target systems. However, the second approach should provide more accurate results by factoring out the performance ratios between two different systems.

Figures 2 and 3 summarize the estimated and measured execution times on the three target systems with both approaches. We can make the following observations.

1. The CRAY-T3E provides about 5 times faster execution than the other two machines. This is due to the fact that we use default optimization in compiling the program in all three systems. It turns out that the default optimization level on Cray-T3E is level 2, while level 0 is default on the other two machines.
2. The estimated execution times based on the model built on the SUN/Enterprise are very accurate up to 32 processors on all three target systems. However, we observe about 15%, -10%, and -3% inaccuracy with the 64-processor CRAY-T3E, IBM-SP2, and SGI-Origin2000 respectively. This can be partially attributed to the fact that the original program was not designed for a large number of processors as can be seen by the increase in execution time beyond 16 processors. Another possible reason for the 15% overestimation on the 64-processor CRAY-T3E system is because of the presence of complex loop structures. Furthermore, due to the diversity of system architectures, the simple performance ratio to extrapolate performance between two systems can create additional misprediction.
3. Our modeling is much more accurate when the performance models are built on each target system. As shown in the above figures, the under/over estimation of the 64 processors can be reduced to about 2.5% in the worst case. The small over-estimation is due mainly to the fact that the potential misprediction factor across different machines is eliminated, which exists in the estimation from the base system. In addition, the execution time equations established from the MATLAB model provide very good estimates of the measured execution times.

4 Conclusion

In this research, a methodology for designing and developing a performance modeling and prediction tool has been suggested, and presented with performance results on three parallel machines. The tool allows users derive performance of applications on different machines with different number of processors without actually executing codes on parallel machines. The strategy is good for quick performance estimation to enable more efficient usage of parallel systems.

The application which we choose as an example for demonstrating the workload characterization method is a simple finite-difference program. Even though the results which are shown from applying the tool to the application have been accurate, it has a limitation to expand the approach to other applications. Other complex programs could not be partitioned into a set of computation and communication regions to characterize workload as we do with the selected program. As people have noticed, a general approach for all programs on all parallel platforms is difficult, and still an open problem.

Acknowledgments

This research was partially supported U.S. Army contract #DACA39-97-K-0037. We also thank Dr. Radha and Dr. Turcotte for providing us access to the three parallel systems at CEWES/MSRC including the IBM-SP2, the CRAY-T3E, and the SGI-Origin2000.

References

1. Bob Cmelik and David Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling," 1994 Sigmetrics, (1994), pp. 128-137.
2. Helen Davis, Stephen Goldschmidt, and John Hennessy, "Multiprocessor Simulation and Tracing Using Tango," Proc. 1991 Int'l Conf. on Parallel Processing, Vol. II, (1991), pp. 99-107.
3. Kivanc Dincer, Zeki Bozkus, Sanjay Ranka, and Geoffrey Fox, "Benchmarking the Computation and Communication Performance of the CM-5," Concurrency: Practice and Experience Vol. 8, No. 1, (1996), pp. 47-69.
4. Roberto Giorgi, Cosimo Antonio Prete, Gianpaolo Prina, and Luigi Ricciardi, "Trace Factory Generating Workloads for Trace-Driven Simulation of Shared-Bus Multiprocessors," IEEE Concurrency, Vol. 5, No. 4, (1997), pp. 54-68.
5. http://www.wes.hpc.mil/documentation/ibmspdoc/ibmsp_faqs.html
6. Manoj Kumar and Kimmung So, "Trace Driven Simulation for Studying MIMD Parallel Computers," Proc. 1989 Int'l Conf. on Parallel Processing, Vol. I, (1989), pp. 68-72.
7. James Lauden, and Daniel Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," 24th Int'l Symp. on Computer Architecture, (1997), pp. 241-251.
8. Lishing Liu and Jih-Kwon Peir: A Performance Evaluation Methodology for Coupled Multiple Supercomputers. Proc. 1990 Int'l Conf. on Parallel Processing, Vol. I, (1990), pp. 198-202.
9. http://www.wes.hpc.mil/documentation/t3edocs/t3e_faqs.html

A Parallel Monte Carlo Algorithm for Protein Accessible Surface Area Computation

Srinivas Aluru*, Desh Ranjan, and Natsuhiko Futamura

¹ Dept. of Electrical and Computer Engineering

Iowa State University, Ames, IA, USA

² Dept. of Computer Science

New Mexico State University, Las Cruces, NM, USA

³ Dept. of Electrical Engineering and Computer Science

Syracuse University, Syracuse, NY, USA

aluru@iastate.edu, dranjan@cs.nmsu.edu, nfutamur@ecs.syr.edu

Abstract. We present sequential and parallel Monte Carlo algorithms for computing the solvent accessible surface area of protein molecules. The basic idea underlying our algorithms is to generate points uniformly at random on the surface of spheres obtained by increasing the van der Waals' radii of the atoms with the van der Waals' radius of the solvent molecule and to test the points for accessibility. We also present an efficient algorithm to compute sphere intersections more efficiently using domain specific knowledge. The expected running time of our sequential algorithm is $O(n + s)$, where n is the number of atoms in the protein and s is the number of points generated. We also provide error bounds as a function of the sample size. Our parallel algorithm can use $O(n)$ processors and provides linear speedup. Computing sphere intersections is common to the various approaches for solving this problem and our algorithm to compute the intersections can be used by them. It takes only $O(n)$ sequential time, which compares favorably with existing algorithms that take $O(n^2)$ worst-case time.

1 Introduction

The accessible surface area of a protein molecule is the cumulative surface area of the individual atoms that is accessible to a solvent molecule. Two atoms of the protein may be close enough that the solvent molecule cannot access the surface area of the atoms completely. The atoms and the solvent molecules are modeled as spheres using their van der Waals' radii. The problem can be further simplified by reducing the solvent molecule to a point and increasing the van der Waals' radii of all the atomic spheres by the van der Waals' radius of the solvent molecule. The problem is now abstracted as: Given n spheres, find the total surface area of the spheres that is accessible, i.e. the surface area that does not lie inside any other sphere.

* Research supported by NSF CAREER Award under CCR-9702991.

Protein ASA computation is used in computational methods for protein folding [9], for estimating the interaction free energy of a protein with solvent, in studies on protein stability and protein-protein interactions. The computation can be decomposed into two parts: Firstly, the set of spheres that intersect each sphere have to be determined. Secondly, we have to find the accessible surface area of each sphere, knowing the spheres that intersect it.

For the first part, a naive algorithm checks all pairwise intersections and takes $O(n^2)$ time. The run-time is significantly reduced in practice by using a decomposition of the space containing the protein molecule and using the decomposition to reduce the number of pairs of spheres examined for possible intersection. However, the worst-case run-time of all the algorithms proposed so far remains $O(n^2)$. For the second part, three different methods have been proposed. One method, due to Lee and Richards [3], is to compute the ASA using numerical integration. Another method, due to Shrake and Rupley [11], is to approximate the spheres by icosahedrons and to use pre-determined sample points for each sphere to approximate the accessible surface area. The third method, due to Richmond [10], is a completely analytical method for directly computing the accessible surface area of a sphere knowing the spheres that intersect it.

Parallel algorithms for ASA computation have been presented in [5,6,12,13]. These algorithms basically consist of finding the sphere intersections in parallel, followed by applying one of the three methods described above for computing the ASA of each sphere. The worst-case run-time of these algorithms for computing the sphere intersections is $O\left(\frac{n^2}{p}\right)$. The problem of computing the individual ASA's for the n spheres is solved by allocating the spheres to the processors and a number of load balancing algorithms and mapping heuristics for this purpose have been studied and experimentally evaluated.

As the spheres in ASA calculations correspond to atoms in proteins, prior information is available which can be used to devise efficient algorithms. For instance, the atoms are of a few types (such as Carbon, Nitrogen etc..) whose radii information is known. Using such domain specific knowledge, we first show that the number of intersections is bounded by $O(n)$ and present a sequential algorithm with expected running time of $O(n)$ to compute these intersections. We also present a parallel version of this algorithm that exhibits linear speedup with respect to the number of processors. For computing the accessible surface area, we present a Monte Carlo algorithm that generates random points on the surface of each sphere and checks which of them are accessible. As the number of generated points increases, the ratio of accessible points to the total number of generated points approaches the fraction of the surface area that is accessible. The scheme has the additional advantage of being highly suitable for parallel computation. We also present error bounds as a function of the sample size.

2 Model of Parallel Computation

We use the *permutation network* as our model of parallel computation. In this model, each processor is allowed to send and receive at most one message during

a communication step. The cost of the communication step is $\tau + \mu l$, where l is the length of the largest message. This corresponds to the assumption that communication corresponding to any permutation can be realized simultaneously. The permutation network model closely reflects the behavior of most multistage interconnection networks.

Our algorithms are stated in terms of the following well-known parallel primitive operations (p denotes the number of processors). For a detailed description and run-time analysis, the reader is referred to [2].

Segmented Parallel Prefix: Segmented prefix computation is a sequence of prefix computations using the same associative operator. Consider $x_{0,0}, x_{0,1}, \dots, x_{0,n_0}, x_{1,0}, x_{1,1}, \dots, x_{1,n_1}, \dots, x_{m-1,0}, x_{m-1,1}, \dots, x_{m-1,n_{m-1}}$, where $\sum_{i=0}^{m-1} n_i = n$ and \otimes is a binary associative operator. We want to compute $s_{i,j}$, where

$$s_{i,j} = x_{i,0} \otimes x_{i,1} \otimes x_{i,2} \otimes \dots \otimes x_{i,j}$$

for $0 \leq i \leq m-1, 0 \leq j \leq n_i - 1$. This operation can be done using one parallel prefix operation and takes $O(\frac{n}{p} + (\tau + \mu) \log p)$ time.

All-to-All Communication: In this operation each processor sends a distinct message of size m to every processor. This operation takes $O((\tau + \mu m)p)$ time.

Transportation Primitive: The transportation primitive performs many-to-many personalized communication with possibly high variance in message sizes. If the total length of the messages being sent out or received at any processor is bounded by t , the transportation primitive performs the communication using two all-to-all communications with a uniform message size of $\frac{t}{p}$ [8].

Sorting: Using sample sort [2] in conjunction with bitonic sort for sorting splitters identified during sample sort, n elements can be sorted on p processors in $O\left(\frac{n \log n}{p} + \tau p + \mu\left(\frac{n}{p} + p \log^2 p\right)\right)$ time for $n > p^2 \log p$.

It should be stressed that the algorithms presented in this paper are applicable to other models of computation as well, and often equally efficiently. This is because we use a few simple communication primitives such as parallel prefix and all-to-all communication.

3 Computing Spherical Intersections

Proteins are chains of amino acid residues. There are 20 different amino acids, which are found in proteins. An examination of the composition of these amino acids reveals that the atoms found in most proteins are carbon, hydrogen, nitrogen, oxygen, phosphorus and sulphur. The van der Waals' radii of these atoms are shown in Table 1.

From the table, we see that the ratio of the largest van der Waals' radius to the smallest van der Waals' radius is a small constant. The atoms do not intersect originally. However, after the radius of each atom is increased by the radius of the solvent molecule, they may intersect. We use the word *atom* to denote an original atom and the word *sphere* to denote the sphere obtained by increasing the radius of the atom by the radius of the solvent molecule.

Table 1. Van der Waals' radii of atoms found in proteins [7]. The three different values shown for carbon are for aromatic carbon, non-aromatic carbon and other carbon, respectively.

atom/molecule	C	H	N	O	P	S	H ₂ O
radius (Å)	1.85/2.0/1.5	1.2	1.5	1.4	1.9	1.85	1.4

Number the atoms (and the corresponding spheres) $1, 2, \dots, n$ and let r_i denote the radius and $C_i = (x_i, y_i, z_i)$ denote the center of the i^{th} atom. Let r be the radius of the solvent molecule. Let $r_{max} = \max\{\max_{i=1}^n\{r_i\}, r\}$ and $r_{min} = \min\{\min_{i=1}^n\{r_i\}, r\}$. Suppose that the ratio of r_{max} to r_{min} is bounded by α , a constant. Consider sphere i . Sphere j intersects sphere i iff $d(C_i, C_j) < r_i + r_j + 2r$, where $d(C_i, C_j)$ denotes the distance between the centers of the two spheres. If sphere j does intersect sphere i , its atom should be completely contained in the sphere of radius $r_i + 2r_j + 2r$ centered at C_i . It follows that all atoms whose spheres may intersect sphere i must lie within the sphere of radius $5r_{max}$ centered at C_i . As the atoms may not intersect, the maximum number of atoms that may lie within this sphere is given by $\frac{4}{3}\pi(5r_{max})^3 / \frac{4}{3}\pi(r_{min})^3 = 125\alpha^3$. The bound can be further improved to $97.3\alpha^3$ by using a well-known fact from sphere-packing that states that no packing of equal-sized spheres can have a density greater than 0.7784 [1,4].

This shows that the total number of spheres that may intersect a given sphere is a constant which in turn implies that the total number of intersections is $O(n)$ where n is the total number of atoms. Note that it is possible to have a configuration of spheres such that every sphere intersects $\Omega(\alpha^3)$ spheres, although this may not happen for protein configurations.

3.1 Sequential Algorithm

Without loss of generality, let $[0, X_{max}] \times [0, Y_{max}] \times [0, Z_{max}]$ be the smallest parallelepiped containing the protein and that X_{max} , Y_{max} and Z_{max} are all divisible by $4r_{max}$. This is the domain relevant to computation of the accessible surface area. Consider an implicit partitioning of the domain into cubes of side length $4r_{max}$. Each cube can be identified with an integer 3-tuple (u, v, w) , where $(u * 4r_{max}, v * 4r_{max}, w * 4r_{max})$ is the corner of the cube having the minimum x , y and z co-ordinates.

The algorithm maintains a hash table T of size n . The idea is to store in $T[k]$, the list of spheres whose centers lie in the cube that hashes to the index k . Hash table collisions are taken care of using chaining in the usual manner. The hash function h takes in a 3-tuple corresponding to a cube and returns an integer in the range $0 \dots (n - 1)$. The table is constructed as follows: For each sphere i find the cube (u_i, v_i, w_i) containing C_i . The hash table entry corresponding to this cube is $T[h(u_i, v_i, w_i)]$. If the cube (u_i, v_i, w_i) is already present in the hash

table, add (C_i, r_i) to the list of spheres contained in this cube. Otherwise, create a new entry for this cube and store (C_i, r_i) in it.

The algorithm then uses this hash table to compute sphere intersections. For each sphere i find the cube containing C_i . Note that the center of any sphere that intersects sphere i must lie in the same cube or in one of the 26 neighboring cubes. The algorithm looks up the hash table entries corresponding to these 27 cubes and checks all the spheres contained in them for intersection with sphere i and builds a list of spheres that do intersect sphere i .

Runtime analysis: Consider the cube (u_i, v_i, w_i) containing C_i . Any atom that has its center within the cube (u_i, v_i, w_i) or one of its 26 neighboring cubes is completely contained in a cube of side length $14r_{max}$ centered at the center of the cube (u_i, v_i, w_i) . It follows that the total number of spheres that have a center in these 27 cubes is bounded by $(14r_{max})^3 / \frac{4}{3}\pi r_{min}^3 \approx 655\alpha^3$. For each sphere, the algorithm accesses the hash table 27 times and for each sphere there are $655\alpha^3$ spheres that are checked for intersection. As the expected time for a hash table access is $O(1)$, the expected running time of the algorithm is $O(n)$.

The constant can be further brought down as follows: Instead of partitioning the domain into cubes of side length $4r_{max}$, we can partition it into cubes of side length γr_{max} , for some constant $\gamma > 0$ and then look up and check for sphere intersections in the relevant cubes. The number of hash table accesses by the algorithm and the maximum number of spheres that may be found in those hash table entries are then both functions of γ . Consider sphere i and the cube (u_i, v_i, w_i) containing C_i . The center of any sphere that intersects sphere i is contained in a cube $(u_i + \Delta_1, v_i + \Delta_2, w_i + \Delta_3)$, where $|\Delta_j| \leq \frac{4}{\gamma}$. Also, the entire atom corresponding to the sphere must lie within the cube of side length $(10 + \gamma)r_{max}$ centered at the center of the cube (u_i, v_i, w_i) . From these two facts, it follows that the number of hash table accesses is $\left(\frac{8}{\gamma} + 1\right)^3$ and the maximum number of spheres that may be found in those entries is bounded by $(10 + \gamma)^3 r_{max}^3 / \frac{4}{3}\pi r_{min}^3 = \frac{3}{4\pi}(10 + \gamma)^3 \alpha^3$.

Choosing a small γ would decrease the maximum number of spheres at the expense of increasing the number of hash table accesses. An optimum value can be found by equating the two and solving the resulting quadratic equation in γ . For example, choosing $\gamma = 2$ reduces the constant to $413\alpha^3$. Once again, the bound can be further improved to $321\alpha^3$ by using the well-known fact from sphere-packing that any packing of equal-sized spheres in 3-space has density at most 0.7784 [1,4]. Note that the number of spheres that the algorithm checks for intersection are expected to be much less than the worst-case bound derived above. However, the formula for the number of hash table accesses is exact. Therefore, it makes sense to choose a value of γ that is favorable to the number of hash table accesses.

3.2 Parallel Algorithm

The parallel algorithm is a step-by-step parallelization of the sequential algorithm described in the previous subsection. For ease of presentation, we assume that n is a multiple of p . Each processor is initially given $\frac{n}{p}$ atoms. The hash table T is partitioned across the processors such that processor j has the portion of the table $T \left[j \frac{n}{p} \dots (j+1) \frac{n}{p} - 1 \right]$.

Each processor scans through the list of atoms assigned to it and prepares entries for hash table insertion. For each entry, the processor which has ownership of this entry can be easily calculated. Each processor prepares $p-1$ messages, one for every other processor. The message to be routed to processor j contains the hash table entries that fall in the portion of the table T allocated to processor j . As each processor has $\frac{n}{p}$ atoms, the total size of all outgoing messages at every processor is $O\left(\frac{n}{p}\right)$. As each processor has $\frac{n}{p}$ hash table entries, the expected total size of all incoming messages at every processor is $O\left(\frac{n}{p}\right)$. The communication is performed using a transportation primitive. The hash table insertions are done locally in each processor. This completes the construction of the hash table.

Next, each processor obtains all the atoms whose spheres might potentially intersect any of the spheres of the atoms assigned to it. This is done as follows: Each processor scans through the list of atoms assigned to it. For each atom, the cube containing the atom is found and the 26 neighboring cubes are determined. All the cubes thus determined are split into $p-1$ groups according to the processors owning hash table entries corresponding to the cubes. The messages are routed to the processors using the transportation primitive. It is clear that the total size of all outgoing messages is bounded by $O\left(\frac{n}{p}\right)$. As for incoming messages, note that a total of $27n$ cubes are generated together on all processors and the expected number of cubes that map to each processor's hash table is $O\left(\frac{n}{p}\right)$. Thus, the total size of all incoming messages for each processor is also bounded by $O\left(\frac{n}{p}\right)$. The requests are satisfied by invoking the transportation primitive. Note that a processor may receive requests for cubes that map to its portion of the hash table but that are not in the hash table. Such requests are ignored as they correspond to empty cubes. Once again, the total outgoing communication size and the total incoming communication size at each processor are bounded by $O\left(\frac{n}{p}\right)$. At this stage, each processor has, for each atom assigned to it, a list of spheres that may potentially intersect the sphere corresponding to the atom. The intersections are checked as in the sequential algorithm.

Runtime Analysis: The parallel algorithm performs expected $O\left(\frac{n}{p}\right)$ work per processor. The communication is performed using three transportation primitives, each of which takes $O\left(\tau p + \mu \frac{n}{p}\right)$ time. The expected running time of this parallel algorithm is $O\left(\frac{n}{p} + \tau p + \mu \frac{n}{p}\right)$. Note that up to $O(n)$ processors can be used.

4 Surface Area Estimation

The accessible surface area of the protein molecule is computed by finding the accessible surface areas of the individual spheres and summing them. We compute the accessible surface area of a sphere as follows: Using the algorithms in Section 3, we have a list of spheres that intersect the sphere under consideration. Generate points uniformly at random on the surface of the sphere. For each point, check if it is inside any of the other spheres that intersect it. The ratio of the points that do not lie inside any sphere to the total number of points tested is an estimation of the accessible surface area of the sphere.

Let S_i be the surface area of sphere i , A_i be its accessible surface area and let A denote the cumulative accessible surface area. Suppose we generate m points uniformly at random on the surface of each sphere, for a total sample size of $s = mn$ points. Let X_i denote the number of points on sphere i that are determined to be accessible. The accessible surface area of sphere i (ASA_i) and the total accessible surface area (ASA) are computed to be

$$ASA_i = \frac{X_i}{m} S_i; \quad ASA = \sum_{i=1}^n ASA_i$$

Note that the X_i 's, ASA_i 's and ASA are all random variables. The probability that a randomly generated point on sphere i turns out to be accessible is $\frac{A_i}{S_i}$. It follows that

$$E[X_i] = m \frac{A_i}{S_i}; \quad E[ASA_i] = A_i; \quad E[ASA] = \sum_{i=1}^n E[ASA_i] = \sum_{i=1}^n A_i = A$$

as desired. As X_i is a binomially distributed random variable and ASA_i is just a multiple of X_i , the variance of ASA_i is given by

$$\text{var}[ASA_i] = \frac{S_i^2}{m} \left(\frac{A_i}{S_i} \right) \left(1 - \frac{A_i}{S_i} \right) = \frac{A_i(S_i - A_i)}{m}$$

By Chebyshev's inequality,

$$Pr \left[|ASA_i - A_i| \geq t \frac{1}{\sqrt{m}} \sqrt{A_i(S_i - A_i)} \right] \leq \frac{1}{t^2}$$

Choosing $t = \delta \sqrt{m} \sqrt{\frac{A_i}{S_i - A_i}}$,

$$Pr [|ASA_i - A_i| \geq \delta A_i] \leq \frac{1}{m \delta^2} \frac{S_i - A_i}{A_i}$$

Recall that ASA_i is the computed approximation to A_i . We would like to ensure with high probability that the error in computing A_i ($|ASA_i - A_i|$) is a small fraction of A_i ($< \delta A_i$). From the previous equation, we require that $\frac{1}{m \delta^2} \frac{S_i - A_i}{A_i}$ be smaller than a user supplied constant $\epsilon > 0$ ($1 - \epsilon$ is usually called the

confidence level). It follows that the required sample size per sphere is $\frac{1}{\epsilon\delta^2} \frac{S_i - A_i}{A_i}$. As expected, this sample size is inversely proportional to ϵ and δ^2 but it is also directly proportional to the ratio of the inaccessible surface area to the accessible surface area on the sphere. If the accessible surface area of a sphere is too small, an impractically large sample size may be required.

Note that if we fix m and ϵ , δ is large if the relative accessible surface area is small. The relative error parameter δ is given by

$$\delta^2 = \frac{1}{m\epsilon} \frac{S_i - A_i}{A_i}$$

and with probability $> 1 - \epsilon$, the absolute error is

$$< \delta A_i = \frac{1}{\sqrt{m\epsilon}} \sqrt{A_i(S_i - A_i)}$$

The maximum value of absolute error occurs when $A_i = S_i/2$, and is $\frac{1}{\sqrt{m\epsilon}} \frac{S_i}{2}$. Note that the corresponding relative error is quite small. In fact, for sphere i , if $A_i < S_i/2$, as the relative error grows larger, the absolute error becomes smaller. We can take advantage of this observation by focusing on the error in computing the total accessible surface area, instead of the error in computing the accessible surface area of each individual sphere.

Since the ASA_i 's are independent random variables,

$$var[ASA] = \sum_{i=1}^n var[ASA_i] = \frac{1}{m} \sum_{i=1}^n A_i(S_i - A_i)$$

By Chebyshev's inequality,

$$Pr \left[|ASA - A| \geq t \sqrt{\frac{1}{m} \sum_{i=1}^n A_i(S_i - A_i)} \right] \leq \frac{1}{t^2}$$

Choosing $t = \delta A \sqrt{\frac{m}{\sum_{i=1}^n A_i(S_i - A_i)}}$,

$$Pr [|ASA - A| \geq \delta A] \leq \frac{1}{m\delta^2} \frac{\sum_{i=1}^n A_i(S_i - A_i)}{A^2}$$

As discussed before, it is desirable if the sample size is a function of only the error tolerance and the confidence level. In order for this to happen, $\frac{\sum_{i=1}^n A_i(S_i - A_i)}{A^2}$ should be bounded by a constant. Therefore, we require

$$\frac{\sum_{i=1}^n A_i(S_i - A_i)}{A^2} \leq \beta$$

for some constant β . As the product of $A_i(S_i - A_i)$ is maximum when $A_i = \frac{S_i}{2}$, we require

$$\frac{\sum_{i=1}^n S_i^2 / 4}{(\sum_{i=1}^n A_i)^2} \leq \beta$$

A sufficient condition for the above equation to be true is

$$\sum_{i=1}^n A_i \geq \frac{1}{2\sqrt{\beta}} \sum_{i=1}^n S_i$$

Therefore, the Monte Carlo method can be applied if the total cumulative accessible surface area is at least a guaranteed fraction of the total surface area and the sample size required varies as the inverse square of this fraction. Fortunately, for most protein molecules, the ratio of the cumulative accessible surface area to the total surface area is not too small. Hence, the Monte Carlo method can be usefully applied.

The algorithm also computes the confidence level of the answer computed. The input to the algorithm is δ , the error tolerance parameter and the sample size per sphere m . It outputs the computed cumulative accessible surface area ASA , and the estimated probability that the error is no more than δA , given by

$$1 - \frac{1}{m\delta^2} \frac{\sum_{i=1}^n ASA_i(S_i - ASA_i)}{\left(\sum_{i=1}^n ASA_i\right)^2}$$

If a better probability is desired, the algorithm can be run with an increased sample size. This Monte Carlo approach to surface area estimation can be used with either a sequential or parallel algorithm. To compute the accessible surface area in parallel, we first use the algorithm of Section 3.2 to compute atomic intersections in parallel. At the end of the execution of this algorithm, each processor has $\frac{n}{p}$ spheres and for each of these spheres, the list of spheres that intersect it. The Monte Carlo approach can be used by all processors in parallel for their $\frac{n}{p}$ spheres. As each spheres intersects with at most a constant number of spheres, this requires $O\left(\frac{s}{p}\right)$ parallel time, where $s = mn$ is the total number of points generated. Finally, the computed partial accessible surface areas on the processors can be combined using a reduce operation in $O((\tau + \mu) \log p)$ time. Thus, the parallel run-time of the surface area estimation is $O\left(\frac{s}{p} + (\tau + \mu) \log p\right)$. Up to $O(n)$ processors can be used with linear speedup provided $p \log p = O(s)$, which is expected to be the case. Combining the parallel algorithms for computing sphere intersections and surface area estimation, the expected running time of the parallel algorithm is $O\left(\frac{n+s}{p} + \tau p + \mu \left(\frac{n}{p} + \log p\right)\right)$. Since $s \gg n$, the algorithm should be communication-efficient, and scale well in practice.

5 Conclusions and Future Research

The main contributions of our research are: 1) to show that the number of sphere intersections in the protein ASA problem is bounded by $O(n)$, 2) optimal sequential and parallel algorithms for computing the sphere intersections, and 3) a Monte Carlo approach to estimating the ASA and the corresponding error analysis. These results are a significant improvement over earlier algorithms,

which had $O(n^2)$ sequential complexity and $O\left(\frac{n^2}{p}\right)$ parallel complexity. Our algorithm for computing sphere intersections can be used with any method for computing protein ASA, including our Monte Carlo method, Lee and Richards method [3], Shrake and Rupley method [11] and Richmond's method [10].

An interesting aspect of our algorithms is the provably optimal run-time and the absence of dynamic mapping or load balancing. Protein ASA computation appears to be an irregularly structured problem, and all the parallel algorithms designed so far use dynamic load balancing strategies. Our result indicates that, contrary to popular belief, it may be possible to design parallel algorithms for solving seemingly ‘irregular’ problems in a regular manner.

In the Monte Carlo approach, we show that the absolute error in estimating the ASA of a sphere for a given confidence level is a function of its ASA and the sample size. This raises an interesting possibility of using different sample sizes for different spheres with the goal of reducing the total absolute error. During sampling, we can compute the estimates of the absolute errors in computing the ASA of various spheres and adapt our sampling strategy so as to generate more points on spheres that help reduce the error most. We are currently investigating such an approach to minimize the error for a given sample size.

References

1. J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, 1988.
2. V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing*, The Benjamin/Cummings Publishing Co., 1994.
3. B. Lee and F.M. Richards, The interpretation of protein structures: estimation of static accessibility, *Journal of Molecular Biology*, 55 (1971) 379-400.
4. J.H. Lindsey II, Sphere-packing in R^3 , *Mathematika*, 33 (1986), 137-147.
5. R.L. Martino, T.K. Yap and E.B. Suh, Parallel algorithms in molecular biology, *Proc. High Performance Computing and Networking* (1997).
6. R.L. Martino, C.A. Johnson, E.B. Suh, B.L. Trus and T.K. Yap, Parallel computing in biomedical research, *Science Vol. 265* (1994) 902-908.
7. L.C. Pauling, *The Nature of the Chemical Bond*, 3rd edition, Cornell University Press, Ithaca, New York, 1960.
8. S. Ranka, R.V. Shankar and K.A. Alsabti, Many-to-many communication with bounded traffic, *Proc. Frontiers of Massively Parallel Computation* (1995), 20-27.
9. F.M. Richards, The protein folding problem, *Scientific American* (1991) 54-63.
10. T.J. Richmond, Solvent accessible surface area and extended volume in proteins – Analytical equations for overlapping spheres and implications for the hydrophobic effect, *Journal of Molecular Biology*, 178 (1984) 63-89.
11. A. Shrake and J.A. Rupley, Environment and exposure to solvent of protein atoms, Lysozyme and Insulin, *Journal of Molecular Biology*, 79 (1973) 351-371.
12. E. Suh, B. Narahari and R. Simha, Dynamic load balancing schemes for computing accessible surface area of protein molecules, *Proc. International Conference on High Performance Computing* (1998) 326-333.
13. E. Suh, B.K. Lee, R. Martino, B. Narahari and A. Choudhary, Parallel computation of solvent accessible surface area of protein molecules, *Proc. International Parallel Processing Symposium* (1993) 685-689.

Parallelisation of a Navier-Stokes Code on a Cluster of Workstations

V.Ashok and Thomas C.Babu

Aerodynamics Division

Aerospace Flight Dynamics Group

Vikram Sarabhai Space Centre, Trivandrum - 695022, INDIA

Abstract An existing Navier-Stokes code (Aeroshape-3D) was parallelised on a cluster of workstations having Message Passing Interface implementation. The parallelisation was done by domain decomposition and incorporating features like parallel communication and load balancing. The parallelisation efficiency achieved for a practical three dimensional problem on a cluster of 8 Dec-Alpha workstations was above 90%. With this capability it is now possible to compute flow field over complex bodies in a reasonable time frame.

1. Introduction

Parallel computing has become very attractive for Computational fluid dynamics (CFD) with the availability of affordable and powerful workstations and advancements in network technology with Message Passing Interface (MPI) becoming a standard of parallel computing. By having a network of such workstations operating in parallel mode, the turn around time to obtain a CFD solution would be reasonable. Also it would be a great boon to the code developer as he will be in a position to see the effect of a modification in the code quite fast if the code is run in a parallel mode. Taking all the above points into consideration it was decided to parallelise the existing Aeroshape-3D code.

2. Parallelisation Procedure

Aeroshape-3D code that is being widely used for solving complex turbulent flow over arbitrary three dimensional bodies [1,2] uses rectangular adaptive Cartesian mesh to define the body and the solution domain. The cell structure is oct tree type when some of the mother cells are split into children to properly capture the body or adapt the solution domain based on flow sensors. The fluxes across the cell faces are computed using an approximate Riemann solver with a min-mod limiter to limit the fluxes across the cell interface [3]. K- ϵ turbulence modeling is used to get the turbulent stresses and the solution is flow adaptive based on any specified flow sensors. The solver is parallelised by the method of domain decomposition i.e the solution domain is split into as much optimal sub-domains as there are number of processors. Here, the solution domain is cut in only one coordinate direction (pipe-wise cutting). While splitting the domain

into optimal sub-domains (Load Balancing) it is ensured that the calculation load in each processor is almost the same. The split results are send to different processors and computation in the parallel mode is started. Since for the calculation of flux vectors of a cell, two of its neighbours in x,y & z directions are required, the first processor will have two additional columns of cells (dummy column of cells) in the end and the last processor will have two additional column of cells in the beginning. The intermediate processors will have two additional column of cells in the beginning and in the end. The cells for which calculations are performed can be called as active cells and cells for which calculations are not performed can be termed as dummy cells. Thus if there are $Nx[0]$ cells in the processor 0 (first processor) in I direction , the last two dummy column of cells $Nx[0]$ & $Nx[0]-1$ are additional cells in the processor 0 which are used to evaluate the flux vector at $Nx[0]-2$ column of cells (last active column) of the processor 0. The values of the flux vectors of these two column of cells $Nx[0]$ & $Nx[0]-1$ which are also present in processor 1 (overlapping column of cells) would be calculated in processor 1 and communicated to the processor 0 for updating the values before the next iteration. The data to be send from one processor to another processor is packed together and send as one single packet of required number of bytes. The receiving processor unpacks the data and allocates the data to the respective cells. The steps involved in parallel computation on 4 processor parallel network is given below

1. Perform calculations for all active cells in the processors 0 to 3.
2. (a) Send flux vector data of last two active columns of cells of processor 0 for which flux vector calculations are done to processor 1. While processor 0 is sending data to processor 1, processor 1 should receive data from processor 0 and allocate the data to first two dummy column of cells 0 & 1 of processor 1. Thus in the next time step calculation the processor 1 would have updated value of flux vectors at dummy column of cells 0 & 1 which are used for the flux vector calculations of second column of cells. Here last two active column of cells of processor 0 and first two dummy column of cells of processor 1 are identical and hence overlapping column of cells which communicate with each other after every iteration.
 - (b) Similarly send flux vector data of last two active columns of cells of processor 1 to processor 2 and allocate them to first two dummy columns of cells of processor 2.
 - (c) Send flux vector data of last two active columns of cells of processor 2 to processor 3 and allocate them to first two dummy columns of cells of processor 3.
3. (a) Send flux vector data of first two active columns of processor 1 (co-column of cells 2 &3)to processor 0. At the same time processor 0 should be ready to receive the data and copy the data to last two dummy columns of processor 0. This data would be used to evaluate flux vector at at last active column of processor 0 in the next iteration.

- (b) Similarly send flux vector data of first two active column of cells (2 &3) of processor 2 to processor 1 and allocate the same to last two dummy columns of cells of processor 1.
 - (c) Send flux vector data of first two active column of cells (2 &3) of processor 3 to processor 2 and allocate the same to last two dummy columns of cells of processor 2.
4. If the iteration number is a perfect multiple of iteration for storing intermediate results store the results in the respective processors.
 5. Collect all the results from other processors to processor 0 and remove the intermediate results in other processors to save disk space in other processors.
 6. Join the results in processor 0 .
 7. Go to step 1 for the next iteration and continue till convergence is obtained.

From the above mentioned procedure for parallel computation it can be noted that communication between the processors is also parallel i.e when all even number processors are sending data, all odd number processors are receiving data and when all odd number processors are sending data, all even number processors are receiving data. This type of parallel communication would greatly reduce the wait time of each processor. It is to be noted that in such a parallel communication set up, the communication time is independent of the number of processors. Also the computational load should be distributed uniformly over all processors to have minimum wait time. Thus proper load balancing and parallel communication would result in an efficient parallel code. The validation of parallel code was done by comparing the solution of flow over a NACA-0012 aerofoil at Mach number 0.9 with serial computation and the solution was found to be identical [4].

3. Results and Discussion

Parallel computation of flow over a jet deflector with firing of solid core motor and 2 strap-on liquid engines was carried out on a cluster of DEC-Alpha Workstations with a 100MbPS fast Ethernet switched network. The surface grid and isometric view of the deflector with core and strap on motors is shown in fig.1. Fig.2 shows the distribution of the solution domain among 4 processors. The solution domain of 100X50X80 mother cells in x,y & z directions are distributed among processors such that each processor has almost equal number of cells. The total number of cells is 485344 of which 144217 are inside the body and hence the balance number of 341127 cells are divided nearly equally among 4 processors and are shown in the table below.

Process_id	Number of Cells in the Solution Domain	I-loop Starting	I-loop Ending	Dummy Cells in I- Direction
0	86094	0	21	Cells 22 & 23
1	85791	22	48	Cells 20 & 21 Cells 49 & 50
2	84774	49	72	Cells 47 & 48 Cells 73 & 74
3	84468	73	99	Cells 71 & 72

The pressure distribution on the centerline of the deflector was obtained by solving 9 simultaneous partial differential equations and the results are compared with experiment [5] as shown in fig.3, which shows a good comparison . Fig.4 shows the plot of speed up obtained on a cluster of DEC-Alpha 500 Mhz Processors. The speed up is expected to be linear as long as the communication time to computing time is very small. For this problem almost linear speed up with more than 90% parallelisation efficiency is obtained up to 8 DEC-Alpha Processors. However the speed up would come down if the number of processors is say more than 25 and then for effective speed up, the parallelisation has to be fine grain.

4. Concluding Remarks

1. Aeroshape-3D code has been successfully parallelised on a cluster of workstations having MPI. The efficiency of parallelisation for a typical real life problem on a cluster of 8 DEC-Alpha workstations is above 90%.
2. Load balancing and Parallel communication in this parallel code enables to have a better speed up.
3. With the availability of more number of processors, parametric study for various complicated 3-D real life problems can be taken up and thus enabling to obtain results for which performing experiments is too difficult or at least reduce the load of experimentation in future.

5. References

1. "Numerical Simulation of Working Processes in Rocket Engine Combustion Chamber". V.N.Gavirilyuk et. al. IAF-93-S.2.463
2. "Nozzle Flow field Analysis With Particular Regard to 3D-Plug-Cluster Configurations". G.Hagemann et.al. AIAA 96-2954.
3. "Aeroshape-3D - Technique and General Description". CAS, Moscow, 1994.
4. "Parallelisation of Aeroshape-3D" V.Ashok, Thomas.C.Babu, VSSC/ARD/TR-011/99
5. "Studies Towards Second Launch Pad (SLP) Design". J.K.Prasad, VSSC/ATF/SLP/GEN/004/99.

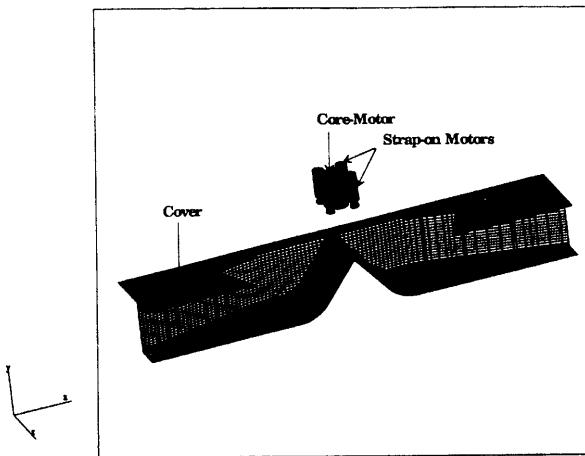


Figure 1: Surface grid of launch deflector

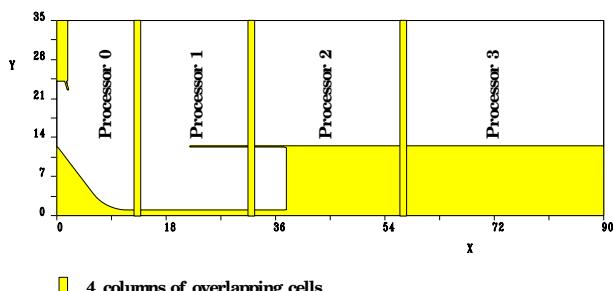


Figure 2: Domain decomposition for 4 processors

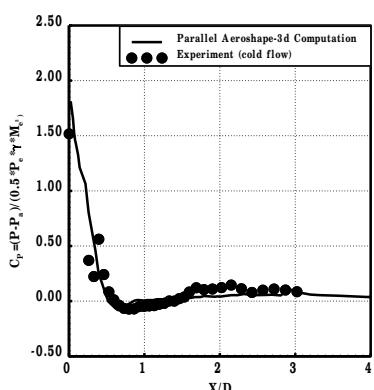


Figure 3: Centre line pressure distribution

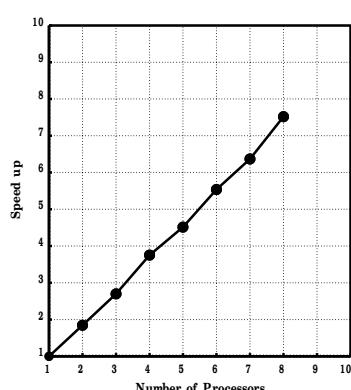


Figure 4: Speed up for ALPHA cluster

I/O Implementation and Evaluation of Parallel Pipelined STAP on High Performance Computers

Wei-keng Liao¹, Alok Choudhary², Donald Weiner¹, and Pramod Varshney¹

¹ EECS Department, Syracuse University, Syracuse, NY 13244
`{wkliao, dweiner, varshney}@syr.edu`

² ECE Department, Northwestern University, Evanston, IL 60208
`choudhar@ece.nwu.edu`

Abstract. This paper presents experimental performance results for a parallel pipeline STAP system with I/O task implementation. In our previous work, a parallel pipeline model was designed for radar signal processing applications on parallel computers. Based on this model, we implemented a real STAP application which demonstrated the performance scalability of this model in terms of throughput and latency. The parallel pipeline model normally does not include I/O task because the input data can be provided directly from radars. However, I/O can also be done through disk file systems if radar data is stored in disks first. In this paper, we study the effect on system performance when the I/O task is incorporated in the parallel pipeline model. We used the parallel file systems on the Intel Paragon and the IBM SP to perform parallel I/O and studied its effects on the overall performance of the pipeline system. All the performance results shown in this paper demonstrated the scalability of parallel I/O implementation on the parallel pipeline STAP system.

1 Introduction

In this paper we build upon our earlier work where we devised strategies for high performance parallel pipeline implementations, in particular, for Space-Time Adaptive Processing (STAP) applications [1, 2]. A modified Pulse Repetition Interval (PRI)-staggered post-Doppler STAP algorithm was implemented based on the parallel pipeline model and scalable performance was obtained both on the Intel Paragon and the IBM SP. Normally, this parallel pipeline system does not include disk I/O costs. Since most radar applications require signal processing in real time, thus far we have assumed that the signal data collected by radar is directly delivered to the pipeline system.

In practice, the I/O can be done either directly from a radar or through disk file systems. In this work we focus on the I/O implementation of the parallel pipeline STAP algorithm when I/O is carried out through a disk file system. Using existing parallel file systems, we investigate the impact of I/O on the overall pipeline system performance. We ran the parallel pipeline STAP system portably

and measured the performance on the Intel Paragon at California Institute of Technology and on the IBM SP at Argonne National Laboratory (ANL.) The parallel file systems on both the Intel Paragon and the IBM SP contain multiple stripe directories for applications to access disk files efficiently. On the Paragon, two PFS file systems with different stripe factors were tested and the results were analyzed to assess the effects of the size of the stripe factor on the STAP pipeline system. On the IBM SP, the performance results were obtained by using the native parallel file system, PIOFS, which has 80 stripe directories.

Comparing the two parallel file systems with different stripe sizes on the Paragon, we found that an I/O bottleneck results when a file system with smaller stripe size is used. Once a bottleneck appears in a pipeline, the throughput which is determined by the task with maximum execution time degrades significantly. On the other hand, the latency is not significantly affected by the bottleneck problem. This is because the latency depends on all the tasks in the pipeline rather than the task with the maximum execution time.

The rest of the paper is organized as follows: in Section 2, we briefly describe our previous work, the parallel pipeline implementation on a STAP algorithm. The parallel file systems tested in this work are described in Section 3. The I/O design and implementation are given in Section 4 and the performance results are given in Section 5.

2 Parallel pipeline STAP system

In our previous work [1], we described the parallel pipelined implementation of a PRI-staggered post-Doppler STAP algorithm. The parallel pipeline system consists of seven tasks: 1)Doppler filter processing, 2)easy weight computation, 3)hard weight computation, 4)easy beamforming, 5)hard beamforming, 6)pulse compression, and 7)CFAR processing. The design of the parallel pipelined STAP algorithm is shown in Figure 1.

The input data set for the pipeline is obtained from a phased array radar and is formed in terms of a coherent processing interval (CPI). Each CPI data set is a 3-dimensional complex data cube. The output of the pipeline is a report on the detection of possible targets. Each task i , $0 \leq i < 7$, is parallelized by evenly partitioning its work load among P_i compute nodes. The execution time associated with task i is T_i . For the computation of the weight vectors for the current CPI data cube, data cubes from previous CPIs are used as input data. This introduces temporal data dependency. Temporal data dependencies are represented by arrows with dashed lines in Figure 1 where $TD_{i,j}$ represents temporal data dependency of task j on data from task i . In a similar manner, spatial data dependencies $SD_{i,j}$ can be defined and are indicated by arrows with solid lines.

Throughput and latency are two important measures for performance evaluation on a pipeline system.

$$\text{throughput} = \frac{1}{\max_{0 \leq i < 7} T_i}. \quad (1)$$

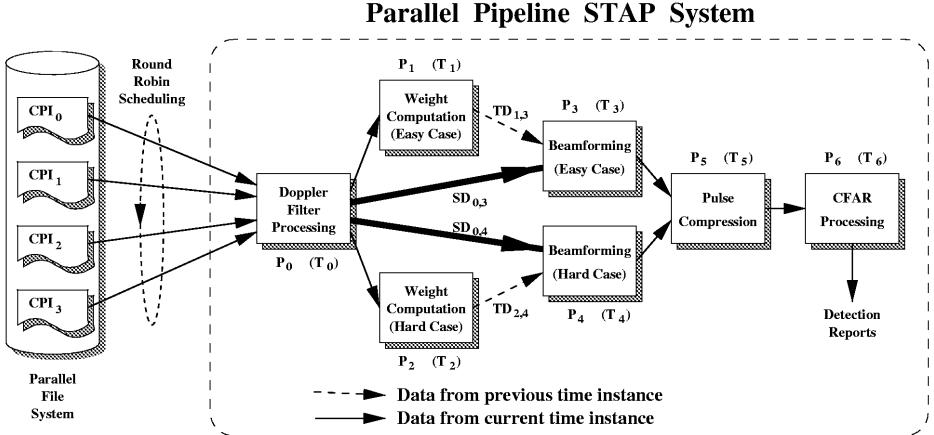


Fig. 1. The parallel pipelined STAP system. Arrows connecting task blocks represent data transfer between tasks. I/O is embedded in the Doppler filter processing task

$$\text{latency} = T_0 + \max(T_3, T_4) + T_5 + T_6. \quad (2)$$

Equation (2) does not contain T_1 and T_2 . The temporal data dependency does not affect the latency because weight computation tasks use CPI data from the previous time instance rather than the current CPI. The filtered CPI data sent to the beamforming task does not wait for the completion of its weight computation. A detailed description of the STAP algorithm we used can be found in [3, 4].

3 Parallel file systems

We used the parallel I/O library developed by Intel Paragon and IBM SP systems to perform read operations. The Intel Paragon OSF/1 operating system provides a special file system type called PFS, for Parallel File System, which gives applications high-speed access to a large amount of disk storage [5]. In this work, two PFS file systems at Caltech were tested : one has 16 stripe directories (stripe factor 16) and the other has a stripe factor of 64. We used the Intel Paragon NX library to implement the I/O of the parallel pipeline STAP system. Subroutine `gopen()` was used to open CPI files globally with a non-collected I/O mode, `M_ASYNC`, because it offers better performance and causes less system overhead. In addition, we used asynchronous I/O function calls: `iread()` and `ireadoff()` in order to overlap I/O with the computation and communication.

The IBM AIX operating system provides a parallel file system called Parallel I/O File System (PIOFS) [6]. There are a total of 80 slices (striped directories) in the ANL PIOFS file system. IBM PIOFS supports existing C read, write, open and close functions. However, unlike the Paragon NX library, asynchronous parallel read/write subroutines are not supported on IBM PIOFS. The overall performance of the STAP pipeline system will be limited by the inability to overlap I/O operations with computation and communication.

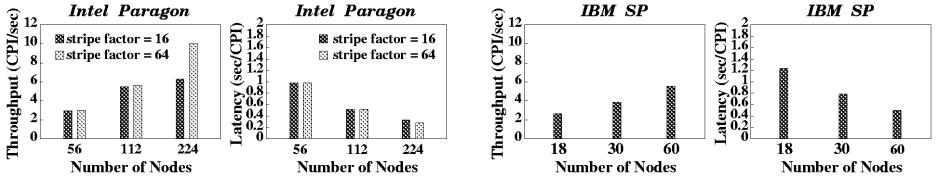


Fig. 2. Performance results for the STAP pipeline system with parallel I/O embedded in the Doppler filter processing task

4 Design and implementation

A total of four CPI data sets stored as four files in the parallel file systems were used on both the Caltech Paragon and the ANL SP. Each of the four CPI files is of size 8M bytes. All nodes allocated to the first task (the I/O nodes) of the pipeline read exclusive portions of each CPI file with proper offsets. Because the number of I/O nodes may vary due to different node assignments to the I/O task, the length of data for the read operations can be different. The read length and file offset for all the read operations are set only during the STAP pipeline system's initialization and is not changed afterward. Therefore, in each of the following iterations, only one read function call is needed. The design for the I/O task implemented in the STAP pipeline system is illustrated in Figure 1.

5 Performance results

In the I/O implementation on the Paragon, the Doppler filter processing task reads its input from CPI files using asynchronous read calls. A double buffering strategy is employed to overlap the I/O operations with computation and communication in this task. Figure 2 shows the timing results for this implementation on the Paragon PFS file system with 16 and 64 stripe directories. Three cases of node assignments to all tasks in the pipeline system are given, each doubles the number of nodes of another. In the case of using the PFS with 16 stripe directories, the throughput scales well in the first two cases, but degrades when the total number of nodes goes up to 224. In this case, the I/O operations for reading CPI data files here become a bottleneck for the pipeline system. This bottleneck forces the rest of the following tasks in the pipeline system to wait for their input data from their previous tasks. On the other hand, both throughput and latency showed linear speedups when using the PFS with 64 stripe directories. In the case with 224 nodes, the I/O bottleneck is relieved by a PFS with more stripe directories.

From the latency results, a linear speedup was obtained. The I/O bottleneck problem does not affect the latency significantly. Unlike the throughput that depends on the maximum of the execution times of all the tasks, the latency is determined by the sum of the execution times of all the tasks except for the tasks with temporal dependency. Therefore, even though the execution time of

the Doppler filter processing task is increased, the delay does not contribute much to the latency.

The timing results for the IBM SP at ANL are also given in Figure 2. Because PIOFS does not provide asynchronous read/write subroutines, the I/O operations do not overlap with computation and communication in the Doppler filter processing task. Hence, the performance results for throughput and latency on the SP did not show the scalability as on the Paragon, even though the SP has faster CPUs.

6 Conclusions

In this work, we studied the effects of parallel I/O implementation on the parallel pipeline system for a modified PRI-staggered post-Doppler STAP algorithm. The parallel pipeline STAP system was run portably on Intel Paragon and IBM SP and the overall performance results demonstrated the linear scalability of our parallel pipeline design when the existing parallel file systems were used in the I/O implementations. On the Paragon, we found that a pipeline bottleneck can result when using a parallel file system with a relatively smaller stripe factor. With a larger stripe factor, a parallel file system can deliver higher efficiency of I/O operations and, therefore, improve the throughput performance. The performance results demonstrate that the parallel pipeline STAP system scaled well even with a more complicated I/O implementation.

7 Acknowledgments

This work was supported by Air Force Materials Command under contract F30602-97-C-0026. We acknowledge the use of the Intel Paragon at California Institute of Technology and the IBM SP at Argonne National Laboratory.

References

1. A. Choudhary, W. Liao, D. Weiner, P. Varshney, R. Linderman, and M. Linderman. Design, Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers. *International Parallel Processing Symposium*, 1998.
2. W. Liao, A. Choudhary, D. Weiner, and P. Varshney. Multi-Threaded Design and Implementation of Parallel Pipelined STAP on Parallel Computers with SMP Nodes. *International Parallel Processing Symposium*, 1999.
3. R. Brown and R. Linderman. Algorithm Development for an Airborne Real-Time STAP Demonstration. *IEEE National Radar Conference*, 1997.
4. M. Linderman and R. Linderman. Real-Time STAP Demonstration on an Embedded High Performance Computer. *IEEE National Radar Conference*, 1997.
5. Intel Corporation. *Paragon System User's Guide*, April 1996.
6. IBM Corp. *IBM AIX Parallel I/O File System: Installation, Administration, and Use*, October 1996.

Efficient Parallel Adaptive Finite Element Methods Using Self-Scheduling Data and Computations

Abani K. Patra, Jingping Long, and Andras Laszloffy

State University Of New York at Buffalo, NY 14260
abani@eng.buffalo.edu

Abstract. Parallel adaptive hp finite element methods (FEM), in which both grid size h and local polynomial order p are dynamically altered, are the most effective discretization schemes for a large class of problems. The greatest difficulty in using these methods on parallel computers is the design of efficient schemes for data storage, access and distribution. We describe here the development of a comprehensive infrastructure *Adaptive Finite Elements Application Programmers Interface* (AFEAPI), that addresses these concerns. AFEAPI provides a simple base for users to develop their own parallel adaptive hp finite element codes. It is responsible for the parallel mesh database, mesh partitioning and redistribution and optionally solution of the large irregularly sparse systems of linear equations generated in these schemes. Dynamic hashing schemes and B-trees are used to store and access the distributed unstructured data efficiently.

1 Introduction

We describe in this paper the development and implementation of efficient algorithms for a highly irregular application – parallel adaptive hp finite element methods for the numerical simulation of physical systems. In such simulations the mesh used to construct the approximate solution is changed in both the number (h is the size of elements) and type of elements (p is the order of approximation used in each element) as the simulation proceeds (see Fig. 1). The changes are made to control the numerical discretization error in the simulation. However, the dynamic nature of the grid (the modifications to the grid depend on the previous solution) creates irregularities in data storage, access and computation that are known at run-time. Further, the adaptive process generates a series of constraints that must be maintained. In a distributed computation these constraints will impose additional inter-processor data consistency requirements.

2 Data Distribution/ Computation Scheduling

Our primary thesis in dealing with this irregularity is to devise means by which the data and computations are *self organizing*. Each object (composed of data and associated computation) is assigned a handle (key) derived from the data

itself. These keys define a simple addressing/ordering scheme for the data and computations. If these keys can be used for data storage, access and distribution of both data and associated computation then the irregularities can be tackled in a simple and efficient fashion. Since the key is derived from the properties of the object, the storage, access and distribution is obvious at the time of object creation and the objects are *self scheduling*.

Following Edwards and Browne [2] “space filling curves” are used for obtaining these keys in our approach. These curves are continuous mappings, $h_n : R \rightarrow U_n$, from the unit interval $R = [0, 1]$ that can completely fill a unit n dimensional hypercube $U_n = [0, 1]^n$ with the images of the unit interval. Algorithms to create these curves are simple and many can be found in the literature [1]. Conversely, given a set of points in U_n one can find a curve that passes through each of these points. The curve can be constructed by implementing the mapping h_n^{-1} i.e. $h_n^{-1}(x_i, y_i, z_i) = \xi_i$ where $\xi_i \in R$ and (x_i, y_i, z_i) are coordinates of the i^{th} point, and sorting the set $\{\xi_i\}$. The sorted set arranges the keys in the order in which the “space filling curve would pass through the points (see Fig 2). Multiplying ξ_i by a large number (10^8) and truncating provides an integer between 0 and 10^8 . This integer will provide a convenient key for each object. Thus given any element or node a key can be computed for it from the coordinates of the centroid of the element and/or the node coordinates. The definition of this key ensures a degree of geometric locality i.e. elements that are close to each other in the physical space are also close to each other in the key space. This key can then be used for storage, access and ordering of the element/node objects.

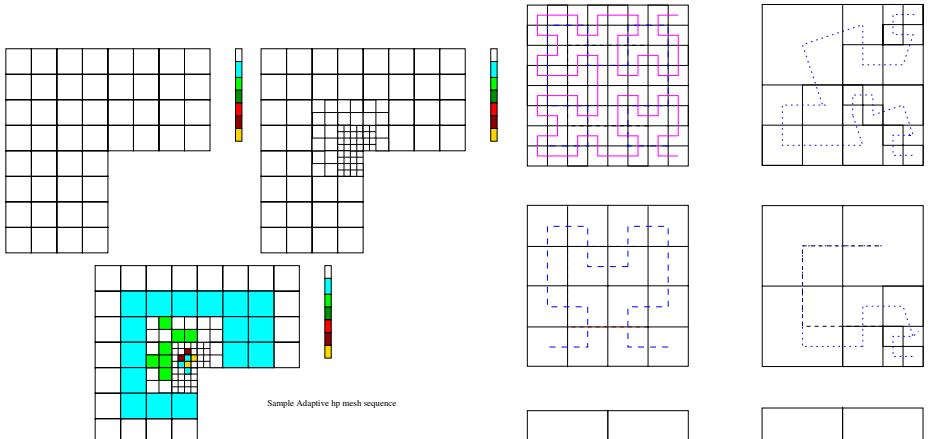
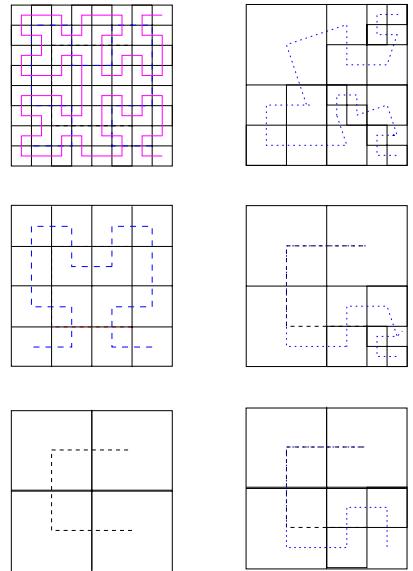


Fig. 1. Evolving grid resulting in irregularity of data access and computing. Colors indicate local polynomial orders.

Fig. 2. Space Filling Curve passing through element centroids



3 AFEAPI: Adaptive Finite Elements Application Programmers Interface

The implementation of this idea results in a simple infrastructure for parallel adaptive hp finite element methods(Fig. 3). The development builds on earlier work [2,7,3,4,5,6] on load distribution obtained by partitioning of adaptive meshes using space filling curves, domain decomposition solvers for such grids and predictive load balancing using *a priori* measures of computational effort.

3.1 Data Structures

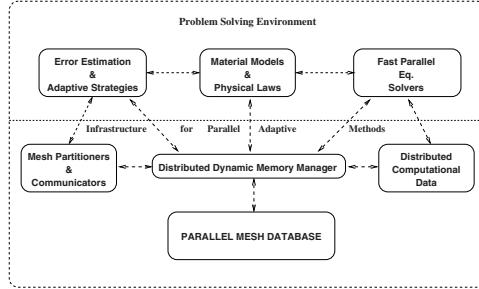
The first component of the infrastructure is a data structure capable of supporting adaptivity i.e. performing the dynamic memory management necessary for element insertion, deletion and modification. The data structure design also needs to integrate support for mesh partitioning and repartitioning and easy inter-processor data movement following adaptivity. Fast data access during the solution process is also a necessary attribute. Data generated and used during a finite element simulation can be classified into persistent grid data and transient computational data. Adopting a owner computes rule reduces the data management problem to that of managing the persistent mesh data only. Two basic data structures are now developed for the grid based data, a hash table based scheme and a B-tree based scheme, that use keys generated by the scheme defined above. The address calculator for the hash table is a simple function like

$$place = \frac{Key - Min.Key}{Max.Key - Min.Key} \times table.size$$

If more than element end up with the same place in the hash table then a small linked list is constructed. In the case of highly adapted meshes with many levels of refinement this linked list can grow very fast. A binary search tree avoids this difficulty but gives up the simplicity and fast access of the hash table. If the objects can be assigned memory that is contiguous in the order of their keys, then the geometric locality of the will cause them to be processed in the same order. Bucket based schemes can be used to achieve this type of contiguous storage assignment. This will lead to memory and storage locality for fast access and cache optimization, a very important consideration in superscalar processors.

3.2 Dynamic Load Balancing

Object distribution via mesh partitioning is automatic as we can achieve a partitioning by recursive bisection (or a simple k-way partition) of the space-filling curve [3]. After each solution cycle predictive load balancing followed by mesh modification is performed using the error estimator as a predictor of computational effort[6]. This greatly mitigates the problems associated with dynamic load balancing and data migration. The mesh modification does require some synchronization of shared data among processors.

**Fig. 3.** Modules of AFEAPI

3.3 Parallel Solvers

The other component of this infrastructure is a multi-level substructuring solver [5] that is integrated with the above data structure. These solvers exploit the natural hierarchical block structure of the higher order finite element methods and efficient geometrically local orderings induced by the space filling curves. We also introduce nested partitionings of the domain to create additional levels of hierarchy. The solver uses a substructuring at lower levels of the hierarchy, switching to a Krylov space type iterative solver with coarse grid preconditioning at an appropriate level [4].

3.4 Customization

The customization of this infrastructure to build application codes for specific simulations comprises essentially of providing a few simple modules that generate an element stiffness matrix when provided with appropriate grid data and produce error estimates by post-processing solution data (see Fig. 3). These modules can often be reused from legacy sequential FORTRAN codes.

4 Results

We present here some tests on the data structure using the SGI Origin2000 and Cray T3E computers. We present test results for element creation, deletion, sequential and random access and data migration on different grids in Table 1 and 2. We instrument the different data management operations and measure time taken to conduct them. The principal purpose of these tests is to guide the tuning of the infrastructure after identifying bottlenecks. Preliminary analysis indicates scalable performance. Code optimization and detailed tests on larger processor sets are currently underway. AFEAPI is available for beta use from <http://wings.buffalo.edu/eng/mae/acm2e>.

Acknowledgements: *The financial support of the National Science Foundation through Grant ASC9702947 is acknowledged. Computer time was provided by NPACI and CCR, University at Buffalo.*

Table 1. Time to traverse full data set and perform ordering of the unknowns on 16 processors of the Cray T3E and the Origin 2000

Ordering	Cray T3E		SGI O2000	
Total dof	hashing	B-tree	hashing	B-tree
8598	0.403s	0.307s	0.230s	0.317s
33328	0.795s	0.662s	0.427s	0.494s
133300	4.340s	4.504s	1.674s	1.892s
534082	61.505s	62.504s	13.927s	14.294s

Table 2. Time to refine elements and propagate constraints on 16 processors of the Cray T3E and the Origin 2000

Refinement	Cray T3E		SGI O2000	
El's refined	hashing	B-tree	hashing	B-tree
1046	0.305s	0.222s	0.095s	0.146s
4184	0.632s	0.819s	0.198s	0.486s
16736	2.984s	4.376s	0.943s	1.194s
66944	30.363s	36.922s	9.594s	8.819s

References

1. H. Sagan, "Space Filling Curves", Springer Verlag, New York, 1995.
2. H. Carter Edwards and J.C. Browne "Scalable Dynamic Distributed Array and Its Application to a Parallel hp adaptive Finite Element Code" Proceedings of POOMA '96: Parallel Object-Oriented Methods and Applications Conference, Santa Fe, New Mexico, February, 1996
3. A. Patra and J. T. Oden "Problem Decomposition Strategies for Adaptive *hp* Finite Element Methods", in *Computing Systems in Engineering*, vol. 6, no. 2, 1995.
4. A. Patra and J. T. Oden, "Computational Techniques for Adaptive *hp* Finite Element Methods", in *Finite Elements in Analysis and Design*. vol 25, 1997, pp. 27-39.
5. A. Patra, "Fast Solvers for Adaptive *hp* Finite Element Methods" **Eighth SIAM Conference on Parallel Processing** Minneapolis, March, 1997.
6. A. Patra and D.W. Kim, "Efficient Mesh partitioning for adaptive *hp* meshes", **XIth Domain Decomposition Conference**, Greenwich, U.K., July 21-24, 1998.
7. S. R. Kohn and S. B. Baden, *A Robust Parallel Programming Model for Dynamic Non-uniform Scientific Computations*, In Proceedings of the Scalable High Performance Computing Conference (SHPCC-94), p p. 509-597. IEEE Computer Society Press, November 1993.

Avoiding Conventional Overheads in Parallel Logic Simulation: A New Architecture

Damian Dalton

Dept of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland.

Abstract. Logic simulation is an important tool in VLSI design. The size of current VLSI circuits is increasing dramatically the computational effort demanded of this design tool. Parallel Processing techniques have reduced computational time. While processing speed is a crucial factor, equally important is the range of delay models that the simulation can support. Unfortunately, some parallel methods limit the accuracy of the delay model. Other parallel methods can only achieve a modest speedup through the use of standard computational mechanisms such as Load balancing and Event-scheduling. Deadlock issues must be resolved in these systems. As the processor numbers increase these tasks grow to the detriment of processing performance. This paper introduces an Associative memory architecture for logic simulation, APPLES, which eliminates the need of conventional support tasks, attains high speedup performance and is capable of simulating complex delay models. The architecture has been implemented as a Verilog model and evaluated theoretically and on various ISCAS-85 benchmarks.

1 Introduction to Parallel Techniques in Logic Simulation

In the testing and verification of digital circuits *Logic Simulation* plays a pivotal position occupying an area where speed of computation is as an important consideration as the accuracy of the results. *Parallel processing* has been investigated as a means to accelerate computational speed. The simulation accuracy is mainly influenced by the gate *delay model*. The simplest delay models assume all gates have a unit delay while complex models have more realistic properties such as inertia and asynchronous rise and fall times.

Two stategies, *Compiled code* and *Event-driven* simulation 1,2 have been employed in parallel logic simulation. The high performance IBM simulation engines, the Yorktown3 and EVE machines 4, 5 have utilised compiled code techniques. However, any compiled code structure is restricted to unit delays and vulnerable to redundant processing.

Event-driven methods eliminate redundant gate evaluation and facilitate complex delay models. Typically, these have an *Event-scheduling/Updating* and *Gate Evaluation* phase. In parallel shared and distributed memory systems, *Synchronous* event MIMD architectures have been evaluated by 6,7. Unfortunately, Load balancing and Global synchronisation contribute significantly to the *Communication overhead*. The best speedup figures are between 3 and 5 on an 8-processor iPSC-Hypercube.

Asynchronous event-driven logic simulation attempts to reduce this overhead. Temporal causality constraints necessitate either a Conservative approach-a strategy to avoid causality errors, or an Optimistic approach-a strategy to recover from causality errors. Conservative methods can terminate in Deadlock, but this has been resolved by deadlock avoidance schemes by 8 and 9. Optimistic methods 10,11, recover from causality errors through Rollback and deadlock recovery. Regardless of the strategy, speedup values peak in the range 20-25, but a value of 10 is far more likely 10, 12. Additionally, each gate evaluation and fan-out updating operation expend up to 250 and 30,000 machine cycles respectively.

This paper presents a prototype architecture specifically designed for logic simulation ; APPLES (Associative Parallel Processor for Logic Event Simulation). *It has been designed to permit complex delay models, reduce or eliminate event related overheads and maximise parallel activity.*

2 Introduction to the APPLES Architecture and Simulation Process

2.1 Synopsis of the APPLES Gate Evaluation/Delay Model Mechanism

In APPLES, a succession of signal values that have appeared on a particular wire over a period of time are stored as a time ordered sequence in a specific word in **Associative memory**, **Associative Array1b (Word-line register Bank)** see Figure(1). For example, a binary value model could store in a 16-bit word, wire values of 16 time unit duration. Gate evaluation proceeds by searching in parallel for appropriate signal values in associative memory. Irrelevant word portions(e.g. only the 5 most recent bits are relevant for a 5-unit gate delay model) are masked out of the search by the memory's **Input** and **Mask** register combination. For a given gate type (e.g. And, Or) and gate delay model (e.g., pure, Inertial etc) there are necessary prerequisites on the structure of the input signals to effect an output change which define an *active* gate. Each prerequisite corresponds to a distinct pattern search in Array1b. A search constitutes a binary **Test**, a match or not.

So that only relevant gate types are selected for a particular search **Tags** are held in **Associative Array1a**. The results of each test are recorded in the **Test-result-register Bank**. Since each gate is assumed to have two inputs (inverters and multiple input gates are translated into their 2-input gate circuit equivalents) tests are *combined* through And or Or operations on bits in word pairs in this bank. This bit-logical combination is specified by the **Result-activator register**.

Every resulting word pair combination is stored as a word in the associative array, **Group-result register Bank**. A search for a distinctive bit pattern in this bank, distinguishes active gates. Successful searches are identified by set bits in the 1-bit column register, **Group-test Hit list**. *This mechanism executes gate evaluation in constant time-the parallel search is independent of the number*

of words. This is an effective linear speedup for this activity. It also facilitates different delay models, since a delay model is simply defined by a set of search patterns.

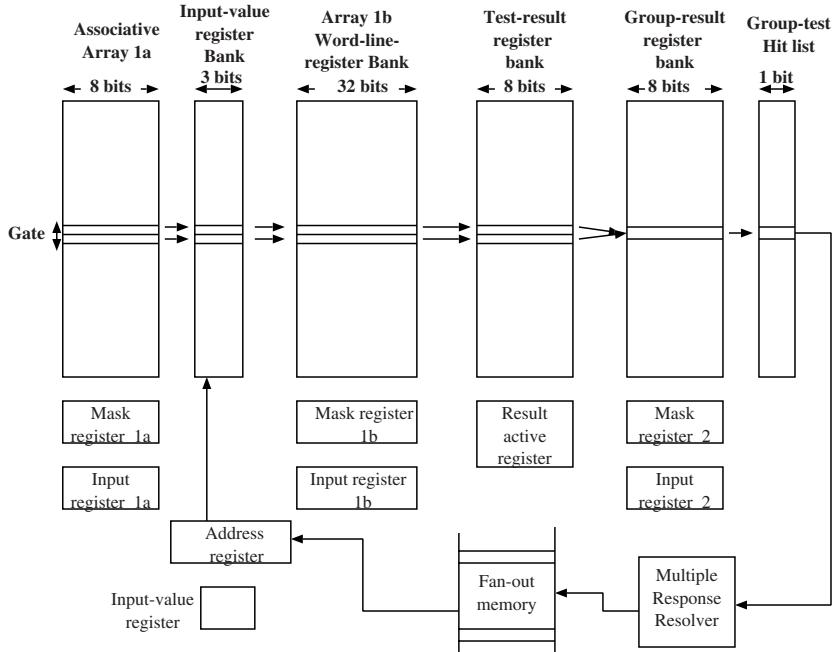


Fig. 1. The APPLES Architecture.

2.2 Synopsis of the APPLES Fan-Out/Update Mechanism

A **Multiple Response Resolver** scans the Group-test Hit list to select active gates. Implemented as a single counter, the resolver inspects the entire list from top to bottom. It stops when a set bit is encountered and then uses its current value as a vector for the active gates' fan-out list. This list has the addresses of the inputs of the fan-out gates in the **Input-value register Bank**. The new logic values of the active gates are written into the appropriate words of this bank. The set bit is then cleared before scanning recommences. By ORing all hit bits, scanning can be prematurely terminated by an all clear condition. Utilising several scan registers accelerates this process. Each scans in parallel and independently an equal size sub-section of the hit list. When all gate types have been evaluated for the current time interval, all signals are updated by *shifting in parallel the words of the Input-value register Bank into the corresponding words of the Word-line register bank*.

2.3 Synopsis of the APPLES Simulation Cycle

Simulation progresses in discrete time units. For any time interval, each gate type is evaluated by applying and combining tests on associative Array1b. This process occupies between 10 machine cycles for the simplest, to 20 machine cycles for the more complex gate delay models, see Figure (2). In general, for 2^N valued logic, N shift operations are required to increment all signal values by one time unit. In the entire simulation cycle, the scan time is the only task which expands with the size of the simulated circuit. Unlike conventional communication overhead, *list scanning* is more conducive to parallelisation.

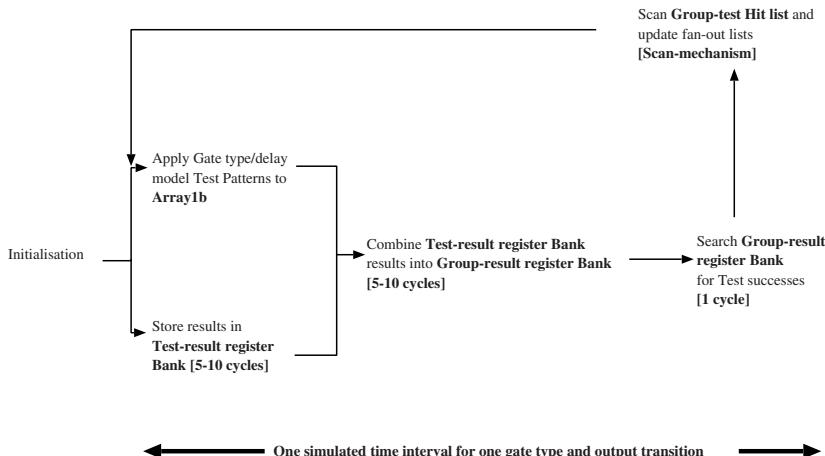


Fig. 2. The Apples simulation cycle.

3 The Inherent Deadlock Avoidance Structure

In parallel event-driven simulation, any system violating any one of the temporal conditions known as the **Input waiting** and **Output waiting** rules 13, will potentially lead to deadlock. All wires are incremented by the smallest timestamp, one discrete time unit. Thus, at the start of every time interval all gates can be evaluated with confidence that the input values are correct; the Input rule is obeyed. The Output rule requiring that signal values are in non-decreasing time order, is observed in each word through the shift operation.

4 Architecture Performance Evaluation

For each gate type, the evaluation time $T_{gate-eval}$ remains constant, typically ranging from 10 to 20 machine cycles. The time to scan the Hit list depends on

its length and the number of registers employed in the scan. N scan registers can divide a Hit list of H locations into N equal partitions of size H/N . Assuming a location can be scanned in 1 machine cycle, the scan time, T_{scan} is H/N cycles. Likewise it will be assumed that 1 cycle will be sufficient to make 1 fan-out update.

When two or more registers simultaneously detect a hit and attempt to access the single ported fan-out memory, a **Clash** occurs. In these circumstances, a semaphore arbitrarily authorises waiting register's accesses to memory. The low activity rate of circuits implies that the probability of three or more simultaneous hits can be ignored. Therefore, the number of clashes during a scan is,

$$\text{No. Clashes} = (\text{Prob of 2 hits per inspection}) \times H/N \quad (1)$$

Assume a uniform random distribution of hits and let Prob_{hit} be the probability that the register will encounter a hit on an inspection. Then (1) becomes,

$$\text{No. Clashes} = {}^N C_2 (\text{Prob}_{\text{hit}})^2 \times H/N \quad (2)$$

Let T_N , be the average total time required to scan and update the fan-out lists of a partition for a particular gate type. Since all partitions are scanned in parallel, T_N also corresponds to the processing time for a N scan register system. Thus the speedup $Sp = T_1/T_N$, of such as system is,

$$T_1/T_N = \frac{T_{\text{gate-eval}} + T_{\text{scan}} + T_{\text{update}}|_{N=1}}{T_{\text{gate-eval}} + H/N + {}^N C_1 (\text{Prob}_{\text{hit}}) H/N + {}^N C_2 (\text{Prob}_{\text{hit}})^2 \times H/N} \quad (3)$$

By differentiating T_N w.r.t N , it can be shown that S_{optimum} is given by,

$$S_{\text{optimum}} = 1/(2.4)\text{Prob}_{\text{hit}} \quad (4)$$

For circuits where the average update time is f_{av} cycles, this will weight all occurrences of Prob_{hit} in Eqt(3). This assumption will be validated with benchmarks.

5 Benchmark Performance

A Verilog model of APPLES simulated 4 **ISCAS-85** benchmarks, **C7552**(4392 gates), **C2670**(1736 gates), **C1908**(1286 gates), **C880**(622 gates) using a unit delay model. Each was exercised with 10 random input vectors over a time period ranging from 1,000 to 10,000 machine cycles. Statistics were gathered as the number of scan registers varied from 1 to 50. The Speedup relative to the number of scan registers is shown in Table 1. Table (1.a) demonstrates that in general the speedup increases with the number of scan registers. The fixed sized overheads of gate evaluation, shifting inputs etc, tends to penalise the performance for the smaller circuits. A more balanced analysis, Table(1.b), is obtained by factoring out all fixed time overheads in the simulation results. This

Table 1. Speedup Performance of Benchmarks

No. Scan Regs					No. Scan Regs				
Circuit	1	15	30	50	Circuit	1	15	30	50
C7552	1	12.5	19.9	24.3	C7552	1	13.6	24.3	29.6
C2670	1	9.7	13.8	15.9	C2670	1	12.5	20.0	25.1
C1908	1	8.4	10.8	11.8	C1908	1	11.8	17.3	20.9
C880	1	7.8	8.3	9.7	C880	1	11.1	12.6	15.9

Speedup					Speedup				
					(excl Fixed Overheads)				
(a)					(b)				

reflects the performance of realistic, large circuits where the fixed overheads will be negligible to the scan time.

Taking the corrected simulated performance statistics, Table (2.a) displays the average number of machine cycles expended to process a gate. APPLES detects intrinsically only active gates. The data takes into account the scan time between hits and the time to update the fan-out lists. As more registers are introduced the time between hits reduces and the gate update rate increases. Clashes happen and active gates are effectively queued in a fan-out/update pipeline. The speedup **saturates** when the fan-out/update rate, governed by the size of the average fan-out list and memory bandwidth, equals the rate at which they enter the pipeline. The validity of the speedup equation, Eqt(3), can be ap-

Table 2. Average No. of Machine Cycles and Fan-out per gate processed

No. Scan Regs					No. Scan Regs				
Circuit	1	15	30	50	Circuit	15	30	50	Av.
C7552	154.6	11.3	6.4	5.2	C7552	0.41	0.35	0.88	0.55
C2670	101.9	8.0	5.1	3.9	C2670	0.52	0.79	1.26	0.86
C1908	86.9	6.8	5.1	3.9	C1908	0.77	1.21	1.32	1.10
C880	49.9	4.9	4.2	3.6	C880	0.16	1.98	1.54	1.22
Av. Cyc/Gate					f_{av}				

(a)					(b)				
-----	--	--	--	--	-----	--	--	--	--

praised from the benchmarks. From the measurements of speedup, Prob_{hit} and the average fan-out cycles for each circuit, the corresponding value for f_{av} was calculated using Eqt(3) is shown in Table(2.b). For a given circuit f_{av} should be constant regardless of the number of scan registers. The values for f_{av} are in accord with the range expected for the fan-out of these circuits. The fluctuations in value across a row for f_{av} , are possibly due to the relatively small number of

samples and size of circuits, where a small perturbation in the distribution of hits in the hit-list can affect significantly the speedup figures.

6 Conclusion

The APPLES gate evaluation process is constant in time. Effectively there is a one to one correspondence between gate and processor (the gate word pairs). This fine grain parallelism allows maximum parallelism in the gate evaluation phase. Active gates are automatically identified and their fan-out lists updated through scanning a hit-list. This scanning is amenable to parallelisation. Multiple scan-registers reduce the overhead time and enable the gate processing rate to be limited solely by the fan-out memory bandwidth. The experimental evidence supports the theory and indicates that substantial speedup values in logic simulation with the APPLES architecture is attainable culminating in a gate processing rate of a few machine cycles. Nevertheless, the design is still a prototype and practical and technical issues concerning its implementation in Xilinx Virtex FPGA technology are being investigated.

References

- Breur et al: Diagnosis and Reliable Design of Digital Systems. Computer Science Press, New York (1976).
- Banerjee: Parallel Algorithms for VLSI Computer-Aided Design. Prentice-Hall (1994).
- Howard et al: Introduction to the IBM Los Gatos Simulation Machine. Proc IEEE Int. Conf. Computer Design: VLSI in Computers. (Oct 1983) 580–583.
- Pfister: The Yorktown Simulation Engine. Introduction 19th ACM/IEEE Design Automation Conf, (June 1982), 51–54.
- Dunn: IBM's Engineering Design System Support for VLSI Design and Verification. IEEE Design and Test Computers, (February 1984) 30–40.
- Soule et al: Parallel Logic Simulation on General purpose machines. Proc Design Automation Conf, (June 1988), 166–171.
- Mueller-Thuns et al: Benchmarking Parallel Processing Platforms: An Application Perspective. IEEE Trans on Parallel and Distributed systems, 4 No 8 (Aug 1993).
- Chandy et al: Asynchronous Distributed Simulation via Sequence of Parallel Computations. Comm ACM 24(ii) (April 1981), 198–206.
- Bryant: Simulation of Packet Communications Architecture Computer Systems. Tech report MIT-LCS-TR-188. MIT Cambridge (1977).
- Briner: Parallel Mixed Level Simulation of Digital Circuits Virtual Time. Ph.D thesis. Dept of El.Eng, Duke University, (1990).
- Jefferson: Virtual time. ACM Trans Programming languages systems, (July 1985) 404–425.
- Soule, Gupta: Characterisation of Parallelism and Deadlocks in Distributed Digital Logic Simulation. Proc 26th Design Automation Conf, (June 1989) 81–86.
- Wong et al: A Parallelism Analyzer for Conservative Parallel Simulation IEEE Trans on Parallel and Distributed Systems. Vol 6 No. 6 June 1995.

Session V-B

Interconnection Networks
Chair: Bhargab Bhattacharya
Indian Statistical Institute

Isomorphic Allocation in k -Ary n -Cube Systems

Moonsoo Kang and Chansu Yu

School of Engineering
Information and Communications University
58-4 Hwa-am, Yu-sung, Taejon, 305-348 KOREA
{kkamo,cyu}@icu.ac.kr

Abstract. Due to its topological generality and flexibility, the k -ary n -cube architecture is actively researched exploring network design tradeoffs as well as characterizing the topology. Processor allocation problem, which has been extensively attempted for hypercubes and meshes, however, has not been addressed for the k -ary n -cube parallel computers. In this paper, we propose *Isomorphic Partitioning*, where a partitioned subcube retains the high order of dimension of the whole system but with smaller size in each dimension. The partitioned subcubes maintain the advantages of the high order architecture. Extensive simulation reveals the effectiveness of the proposed scheme based on the Isomorphic Partitioning.

Index terms - *k -ary n -cube, processor allocation, hypercube multiprocessor, job scheduling, slice partitioning, isomorphic partitioning, performance evaluation.*

1 Introduction

Processor allocation has been studied extensively for directly interconnected parallel computers. Multiple jobs share the topological space of system resources such as processors and memories. Considerable number of allocation algorithms have been proposed for hypercubes [1]-[5] and meshes [6]-[9]. An allocation algorithm dynamically partitions the interconnection topology and the corresponding processors and assigns subsets of processors to the requesting jobs. The problem of the processor allocation assumes importance so as to enable higher system utilization and lower fragmentation of processors by efficiently recognizing a proper subcube.

This paper addresses the processor allocation problem for a k -ary n -cube parallel computers, denoted as Q_n^k , which has k nodes in each of n dimensions. The most commonly used direct networks such as meshes and hypercubes are variants of the k -ary n -cube. The generality and flexibility of the k -ary n -cube make it an active research topic in various aspects [10]-[16]. On the contrary, there have been few works devoted on the processor allocation problem for the k -ary n -cube systems. [17][18] are the only reported results to the authors' knowledge. [17] introduced *EB* (*Extended Buddy*) and *EGC* (*Extended Gray Code*), extended versions of the well-known hypercube algorithms. *k -ary Partner* strategy [18],

which is also an extension of *Partner* hypercube algorithm, enhances the *subcube recognition ability* over EB and EGC. Those strategies, however, limit the job size to be base- k . Jobs requesting different sizes are allocated to one or more partition(s) of base- k and the remaining nodes will be wasted, which is known as *internal fragmentation*. We argue that the job size restriction of their allocation algorithms is inherent with the underlying partitioning mechanism defined as *Slice Partitioning*.

In this paper, we propose a new partitioning mechanism, called *Isomorphic Partitioning* and the corresponding processor allocation algorithms. A k -ary n -cube (Q_n^k) is partitioned into $2^n \frac{k}{2}$ -ary n -cubes ($Q_n^{\frac{k}{2}}$'s). Each of them is divided again into $2^n \frac{k}{4}$ -ary n -cubes ($Q_n^{\frac{k}{4}}$'s) and so on. All of the partitioned subcubes are said to be “*isomorphic*” in the sense that they are all n -cubes. Isomorphic Partitioning eliminates the drawbacks inherent in the Slice Partitioning. By maintaining the higher order architecture also in each partitioned subsystem, it provides shorter inter-node distance. Reducing the average distance is one of the main ideas of building parallel systems of a higher dimension instead of one-dimensional array of nodes.

The rest of the paper is organized as follows. In Section 2, formalism to describe a k -ary n -cube and its partitioning mechanisms are introduced. In Section 3, Isomorphic Strategy is presented. Section 4 is devoted to performance evaluation and comparison of various policies. Conclusions are drawn in the last section.

2 Preliminaries

A k -ary n -cube, denoted by Q_n^k , has k^n nodes each of which can be identified by radix k n -tuple $(a_{n-1}, \dots, a_1, a_0)$ where a_i represents the node's position in the i th direction. Two nodes $(a_{n-1}, \dots, a_1, a_0)$ and $(a'_{n-1}, \dots, a'_1, a'_0)$ are connected if and only if there exists i , $0 \leq i \leq n-1$, such that $a_i = a'_i \pm 1$ and $a_j = a'_j$ for $j \neq i$ if wrap-around links are not included. Given two graphs $A = (V_1, E_1)$ and $B = (V_2, E_2)$, the *cross product* $A \otimes B = (V, E)$ is defined by [12]

$$\begin{aligned} V &= \{(a, b) \mid a \in V_1, b \in V_2\} \\ E &= \{(a, b), (a', b') \mid (a = a' \text{ and } (b, b') \in E_2) \text{ or } (b = b' \text{ and } (a, a') \in E_1)\} \end{aligned}$$

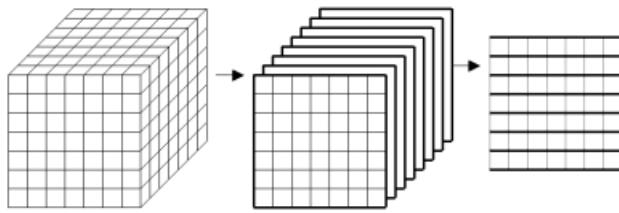
If we let L_k be a graph that has k nodes and k edges, a k -ary n -cube (Q_n^k) can be defined by a *cross product* of n L_k 's [12]¹. That is

$$Q_n^k = \underbrace{L_k \otimes L_k \otimes \dots \otimes L_k}_{n \text{ times}}$$

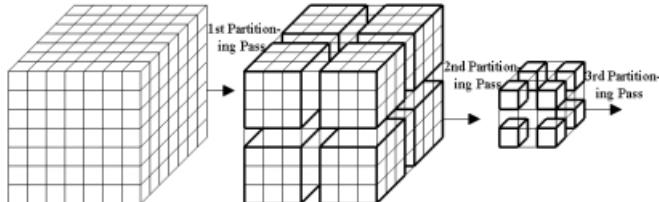
¹ Here, we do not include wraparound edges and the resulting Q_n^k is a mesh. In [12], L_k is a cycle which has the wraparound edge, and the corresponding Q_n^k is a torus.

The *Slice Partitioning* described in Section 1 corresponds to the above construction process in the reverse direction as depicted in Figure 1.a. A higher order cube is partitioned into a number of lower dimensional subcubes.

For simplicity, we deal with 2^k -ary n -cube or $Q_n^{2^k}$ which is defined by the cross product $\underbrace{L_{2^k} \otimes L_{2^k} \otimes \dots \otimes L_{2^k}}_{n \text{ times}}$. Now, we present an alternate way of defining a $Q_n^{2^k}$ based on the *dot product* which multiplies two graphs to produce a larger graph but with the same order of dimension.



(a) Slice Partitioning of an 8-ary 3-cube (8x8x8)



(b) Isomorphic Partitioning of an 8-ary 3-cube (8x8x8)

Fig. 1. Partitioning Mechanism of an 8-ary 3-cube

Given two n -dimensional graphs, $A = (V_1, E_1)$ and $B = (V_2, E_2)$, we define the dot product $A \odot B = (V, E)$ by

$$\begin{aligned} V &= \{(a, b) \mid a \in V_1, b \in V_2\} = \{(a_{n-1}b_{n-1}, \dots, a_1b_1, a_0b_0)\} \\ E &= \{(a, b), (a', b') \mid \exists i, 0 \leq i \leq n-1, \\ &\text{such that } a_i b_i = a'_i b'_i \pm 1 \text{ and } a_j b_j = a'_j b'_j \text{ for } j \neq i\} \end{aligned}$$

If we define an n -dimensional binary hypercube, denoted by B_n , as a graph consisting of 2^n nodes, each of which is represented by a binary n -tuple, $(a_{n-1}, \dots, a_1, a_0)$, where $0 \leq a_i \leq 1$, $Q_n^{2^k}$ can be alternatively defined by a dot product of the binary hypercubes. That is

$$Q_n^{2^k} = \underbrace{B_n \odot B_n \odot \dots \odot B_n}_{k \text{ times}}$$

Consider the corresponding partitioning procedure. Since $Q_n^{2^k} = Q_n^{2^{k-1}} \odot B_n$, a $Q_n^{2^k}$ or $\underbrace{2^k \times \dots \times 2^k}_{n \text{ times}}$ can be “isomorphically” partitioned into $2^n Q_n^{2^{k-1}}$ ’s or $\underbrace{2^{k-1} \times \dots \times 2^{k-1}}_{n \text{ times}}$. We call this as *Isomorphic Partitioning*. Graphical representation of the Isomorphic Partitioning as well as the partitioning pass are shown in Figure 1.b. Refer [19] for mathematical details.

3 Isomorphic Allocation

In this section, we present the Isomorphic allocation algorithm based on the Isomorphic Partitioning. First, we consider how to address a subcube produced by the Isomorphic Partitioning. A node in a 2^k -ary n -cube, $Q_n^{2^k}$, is denoted by an n -tuple $(a_{n-1}, \dots, a_1, a_0)$, where $0 \leq a_i \leq 2^k - 1$, i.e. a_i is a radix 2^k number. We can also denote the node in a full binary representation as

$$(a_{n-1}^{(1)} a_{n-1}^{(2)} \dots a_{n-1}^{(k)}, \dots, a_1^{(1)} a_1^{(2)} \dots a_1^{(k)}, a_0^{(1)} a_0^{(2)} \dots a_0^{(k)}),$$

where $0 \leq a_i^{(j)} \leq 1$. As discussed in Section 2, each of k dot products contributes one binary digit in all dimensions by concatenating the components. We can, therefore, conversely represent it as

$$\underbrace{((a_{n-1}^{(1)}, \dots, a_1^{(1)}, a_0^{(1)}))}_{\text{1st } n\text{-tuple}}, \underbrace{(a_{n-1}^{(2)}, \dots, a_1^{(2)}, a_0^{(2)}), \dots, (a_{n-1}^{(k)}, \dots, a_1^{(k)}, a_0^{(k)})}_{\text{2nd } n\text{-tuple}}, \dots, \underbrace{(a_{n-1}^{(k)}, \dots, a_1^{(k)}, a_0^{(k)}))}_{\text{k-th } n\text{-tuple}}$$

Similarly, a subcube $Q_n^{2^a}$ can be represented by an n -tuple,

$$(a_{n-1}^{(1)} a_{n-1}^{(2)} \dots a_{n-1}^{(k-a)} \underbrace{* \dots *}_{a \text{ times}}, \dots, a_1^{(1)} a_1^{(2)} \dots a_1^{(k-a)} \underbrace{* \dots *}_{a \text{ times}}, a_0^{(1)} a_0^{(2)} \dots a_0^{(k-a)} \underbrace{* \dots *}_{a \text{ times}}),$$

or equivalently,

$$\underbrace{((a_{n-1}^{(1)}, \dots, a_1^{(1)}, a_0^{(1)}))}_{\text{1st } n\text{-tuple}}, \underbrace{(a_{n-1}^{(2)}, \dots, a_1^{(2)}, a_0^{(2)}), \dots, (a_{n-1}^{(k-a)}, \dots, a_1^{(k-a)}, a_0^{(k)}))}_{\text{2nd } n\text{-tuple}}, \dots, \underbrace{(a_{n-1}^{(k-a)}, \dots, a_1^{(k-a)}, a_0^{(k)}))}_{(k-a)-th n\text{-tuple}}$$

It is noted that the first n -tuple identifies one of $2^n Q_n^{2^{k-1}}$ ’s which are generated by the first partitioning pass. For explanatory simplicity, we illustrate the algorithm with an 8-ary 2-cube (8×8 mesh) as in Figure 2. Figure 2.a shows all 64 nodes and the partitioning lines with the Isomorphic Partitioning. It can also be described by a 2^n -ary tree (4-ary tree in this example) with k partitioning passes (3 passes in this case) as in Figure 2.b. Every node in a tree represents a subcube and has its own address within the corresponding partitioning pass. Full address of a node can be identified by concatenating n -tuples obtained by traversing the tree upward to the root. For example, the subcube A ’s address is $((1, 0), (0, 1), (1, 1)) = (101, 011)$. Subcube B is identified

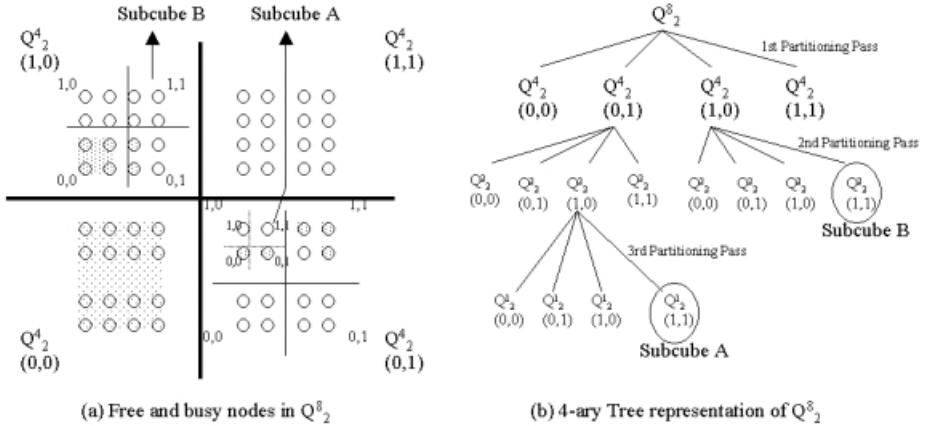


Fig. 2. Representation of the Isomorphic Partitioning of Q_2^8

by $((0,1),(1,1)) = (01*,11*)$. The tree structure is managed by a linked *status bitmap*. A *free list* $F_{a,n}$ is a linked list of free subcubes in the form of $Q_n^{2^a}$ ($\underbrace{2^a \times 2^a \times \dots \times 2^a}_{n \text{ times}}$ or 2^{an} nodes). In the previous example, they are $F_{3,2}$, $F_{2,2}$, $F_{1,2}$ and $F_{0,2}$. For implementation details, see [19].

Isomorphic allocation algorithm handling isomorphic request $(Q_n^{2^a})$ is summarized below. Release algorithm is used when a job finishes its execution. Since it's father node has 2^n children nodes, or buddies, including the freed subcube, it is necessary to check if they all are free so that they can be merged.

Isomorphic Strategy

Request $(Q_n^{2^a})$:

1. If F_{an} is not empty, allocate a subcube in F_{an} to the job.
2. Otherwise, search $F_{(a+1)n}, F_{(a+2)n}, \dots, F_{kn}$, in order until a free subcube is found.
3. If found, decompose it isomorphically until a $Q_n^{2^a}$ is obtained and allocate it to the job. Update the corresponding lists after decomposition.
4. Else enqueue the job to the system queue.

Release $(Q_n^{2^a})$:

1. Insert the released subcube into the list F_{an} .
2. If F_{an} contains all the buddies, merge them to make $Q_n^{2^{a+1}}$ and insert into the list $F_{(a+1)n}$.
3. Repeat step 2 until the corresponding buddies are not available.

The Isomorphic Allocation algorithm is generalized to alleviate the job size limitation by handling non-isomorphic jobs as well as to be applicable to non-isomorphic system. We do not include the extensions here due to the space limit[19].

4 Simulation Model and Results

We conducted a simulation to evaluate and compare the performance of the proposed strategies with other allocation policies. The other simulated schemes are EB (Extended Buddy), EGC (Extended Gray Code) and k -ary Partner algorithms. The buddy allocation policy is employed for the Isomorphic allocation algorithms. Simulation details can be found in exteneded version of this paper[19].

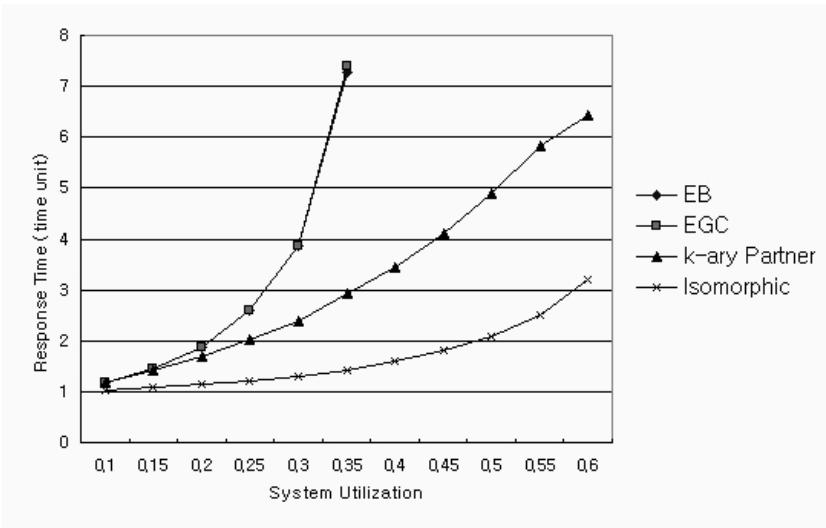


Fig. 3. Mean response time with uniform job size distribution in a Q_3^2

Figure 6 shows the variation of mean response time with respect to input load or system utilization for a 4-ary 3-cube (Q_3^4 or $2^2 \times 2^2 \times 2^2$). The Isomorphic allocation algorithm outperforms other policies in a remarkable performance gap. EB and EGC perform almost the same and they degrade a lot when system utilization reaches beyond 0.3. Equal performance of the EB and EGC policies can be explained by the small system size. As system size grows, EGC will perform better than EB since it has more choices than the proportion. k -ary Partner scheme shows better performance than EB and EGC due to its better subcube recognition ability. But it does not utilize the system well because of the internal fragmentation which is unavoidable with the conventional Slice Partitioning.

Note that the performance gap between the Isomorphic Startegy over k -ary Partner will be greater as system size grows since the internal fragmentation becomes more critical with larger systems. With different sets of simulations, we find that the Isomorphic Strategy is scalable, i.e. it exhibit steady performance irrespective of the system size[19].

5 Conclusion

We have addressed the processor allocation problem for a k -ary n -cube parallel computer. All prior research has been based on the Slice Partitioning, which divides a system topology into a number of low dimensional slices. However, they usually suffer from internal fragmentation of processors due to the job size limitation. In contrast, we propose a new partitioning mechanism, called Isomorphic Partitioning, which divides a system into same dimensional subcubes. Simulation study shows that the Isomorphic Allocation outperforms all existing policies with a remarkable gap. Moreover, the resulting partitions are characterized by the same order of dimension as the whole topology so that they keep the advantages of a high order architecture. Since the Isomorphic Strategy handles semi-isomorphic jobs as well as non-isomorphic architecture, we think it is very attractive for implementation. The Isomorphic Partitioning also provides a new viewpoint in partitioning a system topology. Allocation on other interconnection networks such as meshes can also be improved with the Isomorphic Partitioning. Another future works are average case analysis to predict the number of status bitmaps which are used to implement the Isomorphic allocation algorithm. Also, we will study on the adaptive solution which uses time efficient and space efficient algorithms adaptively depending on load intensity and workload distribution.

References

1. M.S.Chen and K.G.Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," *IEEE Trans. Comput.*, Vol.C-36 pp.1396 -1407, Dec.1987.
2. J.Kim, C.R.Das and W.Lin, "A Top-Down Processor Allocation Scheme for Hypercube Computers," *IEEE Trans. Parallel and Distributed Systems*, Vol.2 pp.20-30, Jan.1991.
3. S.Dutt and J.P.Hayes, "Subcube Allocation in Hypercube Computers," *IEEE Trans. Comput.*, Vol.C-40 pp.341-352, Mar.1991.
4. C.Yu and C.R.Das, "Limit Allocation: An Efficient Processor Management Scheme for Hypercubes," *Proc. Int. Conf. Parallel Processing*, pp.II-143-150, 1994.
5. P.Mohapatra, C.Yu and C.R.Das, "A Lazy Scheduling Scheme for Hypercube Computers," *Journal of Parallel and Distributed Computing*, Vol.27 No.1 pp.26-37, May 1995.

6. B.Yoo, C.R.Das and C.Yu, "Processor Management Techniques for Mesh-Connected Multiprocessors," *Proc. Int. Conf. Parallel Processing*, pp.II-105-112, 1995.
7. S-M.Yoo, H.Y.Yoon and B.Shiraz, "An Efficient Task Allocation Strategies for 2D Mesh Architectures," *IEEE Trans. Parallel and Distributed Systems*, Vol.8 pp.934-942, Sep. 1997.
8. D.D.Sharma and D.K.Pradhan, "Job Scheduling in Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, Vol.9 pp.57-70, Jan. 1998.
9. V.Lo, K.J.Wndisch, W.Liu and B.Nitzberg, "Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, Vol.8 pp.712-726, Jul. 1997.
10. W.Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Comput.*, Vol.39 pp.775-785, Jun.1990.
11. P.Ramanathan and S.Chalasani, "Resource Placement with Multiple Adjacency Constrains in k-ary n-Cubes," *IEEE Trans. Parallel and Distributed Systems*, Vol.6 pp.511-519, May.1995.
12. B.Bose, B.Broeg, Y.Kwon and Y.Ashir, "Lee distance and Topological Properties of k-ary n-cubes," *IEEE Trans. Comput.*, Vol.44 pp.1021-1030, Aug.1995.
13. S.Dutt and N.Trinh, "Are There Advantages to High-Dimension Architectures?: Analysis of k-ary n-cubes for the Class of Parallel Divide-and-Conquer Algorithms," *Proc. ACM ICS'96*, pp.398-406, 1996.
14. K.Day and A.E.Al-Ayyoub, "Fault Diameter of k-ary n-cube Networks," *IEEE Trans. Parallel and Distributed Systems*, Vol.8 pp.903-907, Sep.1997.
15. C.Carrion, R.Beivide, J.A.Gregorio and F.Vallejo, "A Flow Control Mechanism to Avoid Message Deadlock in k-ary n-cube Networks," *Proc. Int. Conf. High Performance Computing*, 1997.
16. D.K.Panda, S.Singal and R.Kesavan, "Multidestination Message Passing in Worm-hole k-ary n-cube Networks with Base Routing Conformed Paths," *IEEE Trans. Parallel and Distributed Systems*, Vol.10 pp.76-96, Jan.1999.
17. V.Gautam and V.Chaudhary, "Subcube Allocation Strategies in a K-ary N-Cube," *Proc. Int. Conf. Parallel and Distributed Computing and Systems*, pp.141-146, 1993.
18. K.Windisch, V.Lo and B.Bose, "Contiguous and Non-Contiguous Processor Allocation Algorithms for k-ary n-cubes," *Proc. Int'l Conf.Parallel Processing*, 1995.
19. M. Kang and C. Yu, "Efficient Allocation Algorithm for *k*-ary *n*-cubes", Technical Report, SICE-TR-003, Information and Communications University, September, 1999.

Unit-Oriented Communication in Real-Time Multihop Networks *

S. Balaji, G. Manimaran, and C. Siva Ram Murthy

¹ Center for Reliable and High-Performance Computing
University of Illinois, Urbana-Champaign, USA.

² Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA

³ Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras 600036, INDIA

balaji@crhc.uiuc.edu, gmani@iastate.edu, murthy@iitm.ernet.in

Abstract. This paper proposes a middleware based solution for supporting unit-oriented communication in real-time networks. A *unit* is characterized by a set of packets with an associated deadline. The problem is to transfer the unit from a given source to destination before its deadline. The existing scheduling disciplines cannot support unit-oriented communication efficiently because they do not exploit the unique features of the unit. In this paper, we propose three bandwidth allocation approaches (naive model, constant bandwidth model, and decreasing bandwidth model) which exploit the unique features of the unit-oriented communication. Based on these approaches, we also make enhancements to well known scheduling disciplines. Our simulation studies show that the proposed middleware based solution improves the call acceptance rate of the unit-oriented calls significantly.

1 Introduction

To guarantee time-constrained communication of packets in a multihop network, real-time channels are to be established with specified traffic characteristics and quality of service (QoS) requirements. The traffic characteristics of a real-time channel include parameters such as maximum packet rate, maximum packet size, and maximum burst size; and parameters such as maximum end-to-end delay and delay jitter, and maximum loss rate constitute the QoS requirements [1]. Throughout this paper, we refer this as *conventional communication*. There are two distinct phases involved in handling real-time channels: channel establishment and run-time packet scheduling. The *channel establishment phase* involves the selection of a route for the channel satisfying traffic characteristics and QoS requirements. During channel establishment, on each link along the path of the channel, a call admission test is performed to check whether enough resources

* This work was done when the authors were at the Indian Institute of Technology, Madras, India.

(such as bandwidth and buffers) are available to satisfy the call requirements. The admission test depends on the scheduling (service) discipline used at run-time [2]. *Run-time scheduling* involves scheduling of packets, at run-time, adhering to the guarantees provided during channel establishment.

The combination of scheduling disciplines and routing algorithms give rise to networks with a wide spectrum of capabilities. Typically, the scheduling discipline and the routing algorithm forms a *service layer* which is responsible for the creation and operation of real-time channels. The applications which require real-time channels can act as an upper layer which uses the services of the service layer. In many applications, the QoS requirements of the applications may not be directly translatable into the QoS parameters supported by the service layer. Hence there is a need for application specific interfaces, called *middleware* which enhance the functionality of the service layer. Thus, both conventional and application specific QoS requirements can be supported by such networks.

Although much work has been done on establishment of real-time channels, with newer scheduling disciplines and routing algorithms, there has been growing efforts in creating middleware based solutions that satisfy application specific QoS requirements. The aim of this paper is to propose a middleware for supporting *unit-oriented communication* in real-time multihop networks. A *unit* is characterized by a set of packets with an associated deadline. The problem is to transfer the entire unit from a given source to destination before its deadline. The unit-oriented communication (e.g., image transfer) differs from conventional communication in the following ways:

- In an unit-oriented communication, the entire unit has a deadline as opposed to individual packets of the unit. Thus, the end-to-end delay needs to be satisfied, by the network, with regards to the unit and not to individual packets.
- The application using unit-oriented communication does not specify the *inter departure time (inter-arrival time)*, X_{min} , of packets during call establishment. Instead, it is left to the network to choose an appropriate value depending on the network condition.
- In most cases, the destination of a unit-oriented communication starts processing the data only when it receives the entire unit.

In [3], a centralized routing algorithm is proposed for the problem, whereas this paper proposes a distributed routing algorithm.

Network Model and Assumptions:

The communication network is modeled as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. An edge $e \in E$ connecting nodes u and v will be denoted by (u, v) .

A call request of an unit-oriented communication channel is a 6-tuple $C = (id, src, dest, p, D, at)$ where, (i) id is a unique identifier which identifies the call request, (ii) src and $dest$ are source and destination nodes of the call, respectively, (iii) p is the number of packets in the unit, (iv) D is the deadline of the unit, and (v) at is the call arrival time.

Each node i assumed to maintain the following data structures. These data structures can be maintained by just monitoring the calls that pass through the node and do not incur any additional communication overhead.

- $NUMHOPS(i, k)$: This denotes the distance (in hop count) between nodes i and k . This value is obtained when a channel was last established between nodes i and k or between nodes i and j passing through node k .
- $LASTDELAY(i, k)$: This denotes the delay between nodes i and k when a channel was last established between nodes i and k or between nodes i and j passing through node k .
- $NUMCHANNELS(i, j)$: For each neighbor node j , the node i monitors the number of channels that are currently using the link (i, j) .
- *Link Utilization*: For each outgoing link j of node i , the link utilization is defined as

$$Util(i, j) = \sum_{k=1}^{NUMCHANNELS(i, j)} \frac{1}{X_{min, k}}, \forall \text{channel } k \text{ passing through link } (i, j). \quad (1)$$

which is the fraction of the total bandwidth of link (i, j) currently being used.

- *Load Parameter L*: This denotes the maximum number of channels that can pass through a node. The Load parameter prevents a node from accepting too many channels, and hence preventing the node getting saturated.

2 Proposed Bandwidth Allocation Approaches

The application layer makes call requests of both unit-oriented and conventional communication requirements, and the service layer together with the middleware provides QoS guarantees to the calls. As part of a conventional call establishment request, maximum as well as average values of bandwidth requirement are usually specified, when a two pass [1] channel establishment scheme is used. During the forward pass, if a node is not able to reserve the maximum bandwidth, it will try to reserve a bandwidth value between the average and maximum. In this process, a downstream node might reserve a lower bandwidth than the upstream nodes. During the reverse pass, the minimum of bandwidth reserved among all the links, along the path, is confirmed and the excess bandwidths are relaxed. In the case of a unit-oriented communication, since the call does not specify the bandwidth requirement, the network has the flexibility to choose a bandwidth value for the call such that the deadline of the unit is met.

In this paper, we propose three bandwidth selection approaches for unit-oriented communication: (i) Naive approach, (ii) Constant bandwidth approach, and (iii) Decreasing bandwidth approach. In the first two approaches, the same amount of bandwidth is allocated in all the links along the path of the channel, whereas the bandwidth could be different in the decreasing bandwidth approach. Since the maximum bandwidth is inversely proportional to X_{min} , hereafter X_{min} will be used to quantify the bandwidth assigned to a channel. In all these approaches, we compute the maximum bandwidth ($1/X_{min}$) requirement for a

unit-oriented call and the average bandwidth ($1/X_{avg}$) requirement is set to this value.

Naive Approach: A naive approach to assign X_{min} is

$$X_{min} = \frac{D - \text{current time} - T_{in}}{p} \quad (2)$$

where, T_{in} is the average time taken to set up a real-time channel.

This approach essentially partitions the entire duration of the unit transfer into equal time intervals and tries to transport one packet per interval. This approach is too rigid and does not exploit the flexibility provided by the unit-oriented communication.

Constant Bandwidth Model: As mentioned in Section 1, the integration of scheduling and routing is very important during channel establishment in order to improve the performance of the network. In our approach, while determining the value of X_{min} of a channel, the nature of the routing approach is also taken into account. If the routing algorithm is flooding based, the value of X_{min} has to be computed for all or subset of links of the source node depending on whether full flooding or partial flooding [4] is used, respectively. If the routing is preferred neighbor based [5], the value of X_{min} is computed only for the link to the preferred neighbor of the source node, which is decided based on the routing heuristic used. The algorithm to determine X_{min} on a link $(source, j)$ of the source node in CBM model is given below.

Algorithm GETXMIN_CBM(*source, dest*) {

1. Let j be the preferred neighbor of source node in the case of preferred neighbor routing, or be any neighbor in the case of flooding based routing.
2. $\text{Remain}(source, j) = 1 - \text{Util}(source, j)$ /* BW available on link $(source, j)$. */
3. $\text{min_}X_{min}(source, j) = 1/\text{Remain}(source, j)$. /* the minimum possible value of X_{min} on link $(source, j)$. */
4. $\text{avg_}X_{min} = \frac{L - \text{NUMCHANNELS}(source, j)}{\text{Remain}(source, j)}$
5. Generate a set of $\text{NUMHOPS}(source, dest)$ values using exponential distribution with mean $\text{avg_}X_{min}$, and store these values in an array called VALUES.
6. Select the maximum value from array VALUES, call it $\text{est_}X_{min}$.
7. $\text{est_Deadline} = \text{est_}X_{min} * (p - 1) + \text{LASTDELAY}(source, dest)$.
8. while ($\text{est_Deadline} > D$ and VALUES array is not empty)
 - Delete $\text{est_}X_{min}$ from VALUES.
 - Select the next $\text{est_}X_{min}$ from VALUES.
 - Recalculate est_Deadline .
9. If ($\text{est_}X_{min} < \text{min_}X_{min}$ or VALUES is empty)
 - then admission test fails on link $(source, j)$; return(failure).
 - Else $X_{min} = \text{est_}X_{min}$; return(X_{min}). }

GETXMIN_CBM uses the past history of the network to get a suitable value for X_{min} . $\text{avg_}X_{min}$ is the average value of X_{min} to be assigned based on the number channels that can use the link $(source, j)$. This is obtained by dividing the limit on the number more channels that can pass through the link (i.e., $L - \text{NUMCHANNELS}(source, j)$) by the fraction of the bandwidth available

($Remain(source, j)$). It may be noted that even when $Remain(source, j)$ is high, the value assigned to X_{min} will depend on $NUMCHANNELS(source, j)$ and L . For example, if $Util(source, j) = 0.5$, $L = 100$, $NUMCHANNELS(source, j) = 50$, then $avg_X_{min} = 100$, i.e., the maximum bandwidth that will be assigned to a new channel will be 0.01 (i.e., 1% of the link bandwidth), even though 0.5 is available. This prevents a channel from taking all the available bandwidth on the link.

Then, a set of random values are generated using exponential distribution with avg_X_{min} as mean. The size of this set is equal to $NUMHOPS(source, dest)$ since this is the upper bound on the number of different X_{min} values possible for a given source and destination based on the current path length. These values, which will lie on either side of avg_X_{min} correspond to the bandwidth which can be allocated to this channel. As we are following the CBM where X_{min} is the same along the path, we go for the maximum of these values (corresponding to minimum bandwidth). Thus, the emphasis is on finding a value which will be acceptable to all nodes that might result in finding a qualified path for the call.

After getting the maximum value (est_X_{min}) from the distribution, the following empirical check is made to see if the channel can be established using est_X_{min} as the inter departure time.

$$est_X_{min} * (p - 1) + LASTDELAY(source, dest) \leq D \quad (3)$$

If the above test is successful and if est_X_{min} is greater than min_X_{min} , then the channel can be set up from the source point of view. If Equation 3 fails, then the next maximum value for est_X_{min} is tried until the array VALUES becomes empty. Finally, the algorithm returns either a value for X_{min} or failure. Note that, satisfying Equation 3 is a necessary condition but not the sufficient condition to set up the channel successfully since the test in Equation 3 is based on local knowledge of the source. Once having found X_{min} , the channel establishment starts with a bandwidth requirement of $1/X_{min}$.

Decreasing Bandwidth Model: The third approach is decreasing bandwidth model (DBM), where a node can allocate less bandwidth than its upstream nodes together with allocation of buffers to compensate the bandwidth mismatch. That is, X_{min} is allowed to change along the path of a channel, thus resulting in different delay bounds guaranteed by different nodes. Buffers are employed at the nodes to store the packets where the bandwidth of incoming link is more than its outgoing link. The difference in delay experienced by a packet between two nodes depends on the amount of time the packet stays in the buffer before being serviced and this difference increases with the buffer size. In a DBM, since buffering plays a crucial part, it is important to determine the exact buffer requirement at each node and also to check the availability of the computed buffer as part of the call admission test. In the modified call admission test, the delay bound at a node, for a unit-oriented call, is the delay experienced by the last packet of the unit.

- **Delay bound:** Let X_{min}^{i-1} and X_{min}^i are the inter departure times at nodes $i - 1$ and i , respectively, where $i - 1$ is the predecessor of i in the path of a channel, and let $X_{min}^{i-1} \leq X_{min}^i$ (by the definition of DBM model).

The delay bound at node i is given by

$$d(i-1, i) = X_{min}^i * p - X_{min}^{i-1} * p \quad (4)$$

which is the worst case delay experienced by the last (p -th) packet.

- **Buffer requirement:** The buffer space required at node i is given by

$$B(i) = \frac{X_{min}^i * p - X_{min}^{i-1} * p}{X_{min}^{i-1}} \quad (5)$$

It can be seen that $B(i)$ increases with time when $X_{min}^i > X_{min}^{i-1}$. Also, it can be seen from Equation 5 that the buffer requirement is zero when $X_{min}^i = X_{min}^{i-1}$.

Determining X_{min} for DBM: The procedure for finding X_{min} for DBM is similar to that of CBM except that the minimum value (highest bandwidth) is chosen from the array VALUES as X_{min} and hence there is no need to iterate (Step 8) over the set of random values generated. Due to space limitations, the algorithm to determine X_{min} in DBM is not presented here.

Admission test at each node in DBM: At each node, to admit a channel on an outgoing link, a check is made to see if the outgoing link can support the incoming link bandwidth allocated to the channel. If so, the bandwidth allocated to the channel on the outgoing link will be the same as that of the incoming link. Otherwise, a lower bandwidth is allocated and an additional check is made to see whether sufficient buffer is available to compensate the bandwidth mismatch. From then onwards, the bandwidth requirement of the call will be the bandwidth guaranteed by the outgoing link. For both CBM and DBM, the admission test performed at intermediate nodes corresponds to the scheduling discipline used.

Admission test at the destination in CBM and DBM: At the destination, the end-to-end delay offered by the network, say d , is known and the following check is made to see whether all the packets of the unit can be transferred before the unit deadline.

$$X_{min} * (p - 1) + d \leq (D - t - (T_{fp} + T_{rp}) - T_{delay}) \quad (6)$$

where, the term $X_{min} * (p - 1)$ denotes the time at which the last packet of the unit will be transmitted at the source, relative to the transmission of the first packet of the unit, based on the X_{min} guaranteed at the source; t is the current time; T_{fp} is the time taken for forward pass; T_{rp} is the time taken for reverse pass; and T_{delay} is the time between the initiation and establishment of the call.

If the above condition is satisfied, the call is accepted, otherwise, the call is rejected. But, in a flooding based routing, it is not necessary to reject the call if this condition is not satisfied. Alternately, one can use dispersity routing wherein the bandwidth requirement of a channel is split across multiple sub-channels in order to meet the delay bound.

3 Simulation Studies

In our simulation studies, we have implemented unit-oriented middleware over the Delay-EDD [1] (scheduler based) and Stop&Go [6] (rate based) scheduling

disciplines and flooding as the routing algorithm with traffic dispersion capability. The following three combinations are considered: (i) Delay-EDD with CBM to get X_{min} , (ii) Stop&Go with one frame and CBM, called Stop&Go(CBM), and (iii) Stop&Go with 4 frames and DBM, to get X_{min} along the path, called Stop&Go(DBM). The naive approach was also implemented over Stop&Go with 4 frames.

The studies were carried out on ARPA network (21 nodes, 26 links) topology. In our studies, both unit-oriented and conventional call requests were generated. The parameter *Traffic Mix* determines the fraction of the total calls generated were unit-oriented calls. The simulation parameters are listed in the table below.

Simulation parameter	Value taken when varied (fixed)
<i>Unit Size</i>	100-1000 (1000)
<i>Arrival Rate</i>	0.01-0.1 (0.01)
<i>Load Parameter</i>	10-100 (100)
<i>Traffic Mix</i>	0.1-0.9 (0.5)
<i>Deadline Max</i>	1000-20000 (20000)
<i>Buffer Max</i>	1000-20000 (20000)

For Stop&Go(1), the frame size was taken as 5. For Stop&Go(DBM), the frame sizes were taken as 5, 10, 20 and 40. The maximum hop count of a channel was set to 15. This is to prevent a convoluted path from getting established. For conventional real-time channels, the duration of the channel was set to a maximum of 75 time units. The bandwidth requirements of these channels were taken in the range 0.01 to 0.1. The performance metric is the *average call acceptance rate (ACAR)* defined as the ratio of number of unit-oriented calls accepted to the number of unit-oriented calls arrived in the network.

Effect of Call Arrival Rate: From Figure 1, the ACAR decreases as the call arrival rate increases. From the figure, three key observations can be made:

- CBM and DBM approaches perform better than the naive approach. This is because these approaches are efficient in selecting a proper value of X_{min} (i.e., bandwidth) for the unit-oriented calls.
- Among CBM and DBM, DBM performs better than CBM for the same scheduling discipline (refer plot corresponding to Stop&Go). This is due to the fact the DBM makes use of buffers for compensating bandwidth mismatch between incoming and outgoing links of a channel.
- Among Delay-EDD and Stop&Go, Delay-EDD performs better than Stop&Go for the CBM model. The same is true for the DBM model (not shown in the figure). This is because Delay-EDD can guarantee a lower delay bound, at a node, than Stop&Go for the same bandwidth allocated.

Effect of Buffer Capacity of a node: From Figure 2, it can be seen that the ACAR increases with increasing buffer size. The ACAR remains almost constant beyond a point (when buffer size is 10). This is because beyond this point, the unit deadline is the bottleneck as opposed to the bandwidth or buffer availability. It can also be seen that Stop&Go(DBM) varies much more than

Stop&Go(CBM). This is because of the same reason mentioned in item 2 for Figure 1.

Effect of Unit Size: When the unit size increases, the ACAR decreases (Figure 3). This is natural because the time at which the last packet of a larger unit reaches the destination is greater than that of the last packet of a smaller unit.

4 Conclusions

In this paper, we have proposed a middleware based solution for supporting unit-oriented communication by proposing three bandwidth allocation approaches (naive, CBM, and DBM). To study the impact of these approaches, we have enhanced the call admission test of Delay-EDD and Stop&Go scheduling disciplines. The simulation studies show that the enhanced admission control improves the call acceptance rate significantly. The following are some of the observations from the simulation studies: (i) DBM performs better than other two approaches and (ii) scheduling discipline influences the performance significantly (from our studies, Delay-EDD offers better ACAR than Stop&Go). The unit-oriented multicast communication would be an interesting area of further research.

References

1. D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE JSAC*, vol.8, no.3, pp.368-379, Apr. 1990.
2. H. Zhang, "Some service disciplines for guaranteed performance in packet-switching networks," *Proc. IEEE*, vol.83, no.10, pp.1374-1396, Oct. 1995.
3. N.S.V. Rao and S.G. Batsell, "QoS routing via multiple paths using bandwidth reservation," in *Proc. IEEE INFOCOM*, pp.11-18, 1998.
4. K.G. Shin and C. Chou, "A distributed route-selection scheme for establishing real-time channels," in *Proc. High Performance Networking*, pp.319-330, 1995.
5. G. Manimaran, H.S. Rahul, and C. Siva Ram Murthy, "A new distributed route selection approach for channel establishment in real-time networks," *IEEE/ACM Trans. Networking*, Oct. 1999.
6. S. J. Golestani, "Stop and Go queueing framework for congestion management," in *Proc. ACM SIGCOMM*, 1990.

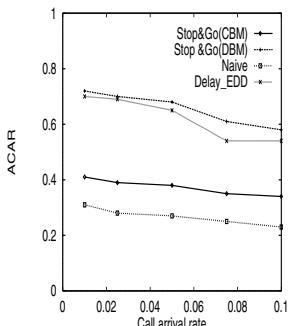


Fig.1. Effect of call load

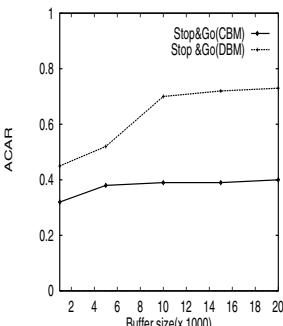


Fig.2. Effect of buffer size

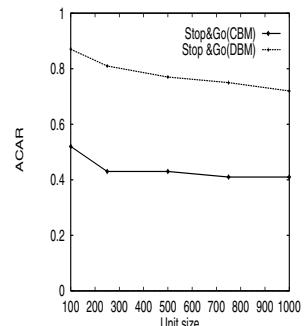


Fig.3. Effect of unit size

Counter-Based Routing Policies

Xicheng Liu, Yangzhao Xiang, and Timothy J. Li

Motorola-ICT Joint R & D Laboratory, Chinese Academy of Sciences
Beijing 100080, P. R. China
Email: xcliu@cti.com.cn

Abstract. Routing policy, also known as selection function, is an important constituent of adaptive routing algorithm for interconnection networks. However, relatively very small research effort has been put on it until now. In this paper, we analyze how routing policy affects network performance. It is disclosed that routing policy actually works as a traffic scheduler. Five generic routing policies are proposed independent of particular routing algorithm. They explicitly schedule the real-time network traffic based on traffic measurement. Counters are used as cost-effective measurement facilities. To standardize the design, constituents of the counter-based routing policy (CRP) are formally specified. Simulations covering extensive traffic patterns exhibit potentials of CRP in maximizing the network performance.

1 Background

For an interconnection network, an adaptive routing algorithm actually consists of two parts, a routing function and a selection function. The routing function supplies a set of routable paths, and the selection function decides one from them. The selection function is more generally called routing policy [1]. Though a great deal of research has been done in designing adaptive routing algorithms, much less attention was put on the routing policy. However, some previous work has indicated that the overall routing performance may owe much to it.

Badr proved that the diagonal routing policy is optimal for 2D meshes [1]. However, for wormhole-routed networks, this scheme is not as advantageous in practice as it seems theoretically because it tends to cause more blocking.

Feng evaluated commonly used routing policies by extensive simulations [2], and observed that none of them is always best for all traffic patterns. By making a selection function to balance the channel utilization, Rao shows that the routing policy can directly affect the network traffic [4].

There is also a reservoir of literature in queueing theory related with the routing policy [3]. There routing policy is of a multi-server queueing problem. It is suggested that if all queues have equal service rates, the message should join the shortest queue. Otherwise, it should join the queue with the minimal expected value of unfinished work. The later is called the minimum expected delay policy (MED). However, this policy has not been practically tested in the context of interconnection networks.

In this paper, we will analyze how routing policy affects network performance, and design a family of generic routing policies independent of specific routing algorithm.

2 Routing Policy as a Traffic Scheduler

In practical interconnection networks, traffic flows are combined to share some common channels. So the traffic is mapped onto a resource-limited network. Thus network topology imposes a first constraint on the traffic distribution. Different routing algorithms provide different set of alternative paths for messages. These would lead to different distributions of traffic. Constraints from the network topology and the routing algorithm are static. Their influence on traffic does not vary with the real-time behavior of traffic. We call them the traffic shaper.

Routing policy decides one definite path for routing from several alternatives. It directs the traffic at packet by packet level. Different strategies may produce very different real time traffic distributions. So it serves as a traffic scheduler. In the context of virtual channels, there is another element contributing to the scheduler, namely, the channel arbitration, which schedules the traffic at flit or packet by packet level.

Therefore, generated traffic, even if it is uniform, may become uneven after constrained with the topology, routing algorithm, routing policy, and channel arbitration. We call original traffic the source traffic, and resulting traffic the object traffic. For any practical interconnection network, above factors constitute a traffic mapping paradigm to shape the source traffic to the object traffic.

What determines the network performance is actually the balance of the object traffic rather than the source traffic. When the topology and the routing algorithm are determined, the routing policy serves as a main engine to schedule the traffic. We can expect it work effectively.

3 Counter-Based Routing Policies

Now that routing policy affects the network performance by scheduling the real-time traffic, we base its design on measurement of the traffic rather than some assumed models. Counters serve as cost-effective measurement facilities. Path selection for next hop is based on the measured parameters. We now formally introduce the counter-based routing policy (CRP).

DEFINITION 1 An *interconnection network* I is a directed graph $I = G(N, C)$. The vertices N represent routers and the arcs C represent communication channels.

DEFINITION 2 A *routing function* R is a mapping defined on $I, R: N \times N \rightarrow P(C)$, where $P(C)$ is a power set of C . For any pair of nodes $(n_c, n_d) \in N \times N$, it provides a set of possible output ports or channels $(c_1, c_2, \dots, c_m) \in P(C)$ at n_c for a message staying there whose destination is n_d . Here, m is less than the maximum number of output ports at n_c .

DEFINITION 3 A *routing policy* S is set of selection rules realizing a mapping $S: P(C) \times V \times F \rightarrow C$. It selects one output port $c \in C$ from the set of alternatives $(c_1, c_2, \dots, c_m) \in P(C)$ supplied by R . The selection will refer to the router status and the traffic behavior at that time. $F = \{FREE, BUSY\}$ represents the status of an output port, available or not. V is a set of parameters representing the real-time traffic status S needed to make a decision. This information is represented as a vector $V = (v_1, v_2, \dots, v_p)$, where $p \geq 1$. An element of it $v_i \in V$ may be a directly measured parameter, but generally it is the result of an algorithm using directly measured parameters as the input. This will be described in definition 4.

DEFINITION 4 A *data extractor* X is a transform $X: U \rightarrow V$, where U represents directly measured traffic data. The transform maps raw data into featuring parameters. $U = (u_1, u_2, \dots,$

u_q , where $q \geq 1$, is also a vector, but may have a different number of elements from V , i.e., $q \neq p$. In the context of multicomputer interconnection networks, the data extractor should be a very simple transform consuming little processing power.

DEFINITION 5 A *counter* is a facility with a value and two operations to update its value. When an input comes, it increases by 1. When a resetting signal is triggered, it is set to the initial value.

From above definitions we see that there are three elements in a CRP. They are a set of counters, a data extractor, and a set of selection rules. Counters are cost-effective traffic meters. A data extractor is a preprocessor to compact the measured data. Selection rules are the actual policies to make path choices based on the preprocessed traffic parameters. We now design five counter-based routing policies balancing real-time traffic in different ways.

Maximum free interval routing policy (MaxFI). This routing policy is that, for a set of alternative output ports $(c_1, c_2, \dots, c_m) \in P(C)$ supplied by R , the port which has been free for the longest time will be selected.

The routing policy requires a counter u for each output port c . It counts the port's free interval from the leaving time of last message. The data extractor for this routing policy is $V = U$. So it is actually omitted. The selection rule for this routing policy is to find the output port with maximum free interval. It is expressed as formula (1). $u_{(i)}$ is the counter for output port c_i . $a_{(i)}$ is the availability of output port c_i . If there are multiple output ports meeting the requirements, we use the first one reached in the search.

$$c = \{c_i \mid u_{(i)} \geq u_{(j)} \cap i \in M, \forall j \in M \cap j \neq i\}, \text{ where } M = \{i \mid a_{(i)} = \text{FREE} \cap i \leq m\} \quad (1)$$

Minimum waiting time routing policy (MinWT). According to this policy, the port having undergone least congestion will be selected. The congestion is measured with the total waiting time of all messages passing through the port. The routing policy also needs a counter u for each output port c . u accumulates messages' waiting time at the port. If there are multiple messages in the output queue, waiting time should be accumulated for all of them. The data extractor for this routing policy is $V = U$. The selection rule for this routing policy is to find the output port with minimum waiting time. It is expressed as formula (2).

$$c = \{c_i \mid u_{(i)} \leq u_{(j)} \cap i \in M, \forall j \in M \cap j \neq i\}, \text{ where } M = \{i \mid a_{(i)} = \text{FREE} \cap i \leq m\} \quad (2)$$

Minimum message number routing policy (MinMN). In this routing policy, the number of messages passing through an output port is used as a traffic indicator. The ports with larger message numbers are more crowded, and are avoided to use if possible. This is reasonable when all output channels have equal bandwidth.

A counter of messages is needed for each output port in this routing policy. Similar to the MinWT policy, MinMN selects the output port with least counter value.

Minimum channel utilization routing policy (MinCU). In this policy, the utilization of each output port of a router is recorded. The output port with least channel utilization is selected. In this way we explicitly balance the channel utilization.

Channel utilization equals the usage time of a port divided by elapsed time. Since elapsed time for all ports is the same, we use the port usage time for comparisons. So this policy needs a counter u for each output port c to accumulate its usage time. The data extractor for this routing policy is $V = U$. The selection rule is as formula (2).

Minimum expected delay routing policy (MED). The MED routing policy chooses a port with least expected delay for a message. Because the exact delay can not be predicted precisely, we approximate this policy by using the average time of a message spent on the port. It equals to the average transmission time plus the average waiting time of a message on a port. Two counters u_n and u_r are needed for each output

port in this routing policy. u_i counts the number of messages passing through the port. u_i accumulates usage time of the port. Thus, $U = (u_i, u_j)$. The data extractor of this policy is $v = u_i/u_n$. v is the featuring parameter. We select the port with least v value from the set of ports provided by R . The selection rule is as formula (3). $v_{(i)}$ is the featuring parameter of port c_i .

$$c = \{c_i \mid v_{(i)} \leq v_{(j)} \cap i \in M, \forall j \in M \cap j \neq i\}, \text{ where } M = \{i \mid a_{(i)} = \text{FREE} \cap i \leq m\} \quad (2)$$

4 Results and Conclusions

Three traffic patterns are used in the simulations, namely, uniform, hotspot, and transpose permutation traffic. Simulations are for 16×16 2D meshes.

Five CRPs and two well-known good routing policies are compared, namely, the zig-zag and the random routing policies, in terms of network delay and network throughput. The simulation results are shown in figure 1 - figure 6. For all traffic patterns the saturate throughputs for CRPs are around 40%. Below saturation, all CRPs produce fairly low network delays. This performance is among the best for single flit buffer wormhole routing.

For network throughput, CRPs perform better than other two policies. Particularly, MED excels all. For network delay, MaxFI performs the best under uniform traffic, and MED exceeds others much under transpose traffic.

Figure 7 - figure 8 show practical channel utilization maps for MED. The data are for unidirectional channels along the +X direction. Normalized network load for the simulations is 40%. We can see that not only uniform traffic is very well balanced at running time, but also hotspot traffic is decentralized and shaped into plateau. Thus bottleneck is alleviated.

In summary, counter-based routing policies provide cost-effective means to explicitly schedule real-time traffic. They balance network traffic so that the network can bear higher load while keeping the average message delay the same or even shorter.

References

1. Badr, H.G., Podar, S.: An Optimal Shortest-Path Routing Policy for Network Computers with Regular Mesh-Connected Topologies. *IEEE Transactions on Computers*. 10 (1989) 1362-1371
2. Feng, W., and Shin, K.G.: Impact of Selection Functions on Routing Algorithm Performance in Multicomputer Networks. *Proceedings of the 11th Annual Conference on Supercomputing* (1997)
3. Lui, J.C.S., Muntz, R., Towsley, D.: Bounding the Mean Response Time of the Minimum Expected Delay Routing Policy: an Algorithm Approach. *IEEE Transactions on Computers*. 12 (1995) 1371-1382
4. Rao, R.: Utilization Imbalance in Wormhole Routed Networks. Master's Thesis. Ohio State University (1994)

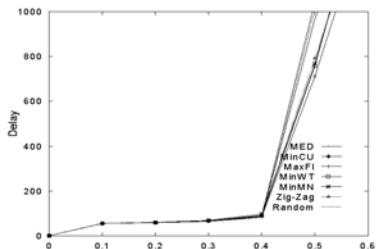


Fig. 1. Net. delay for unif. traff.

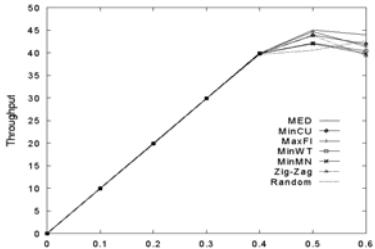


Fig. 2. Net. throughput for unif. traff.

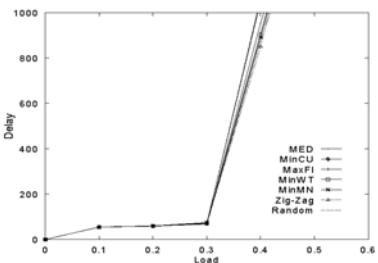


Fig. 3. Net. delay for hotspot traff.

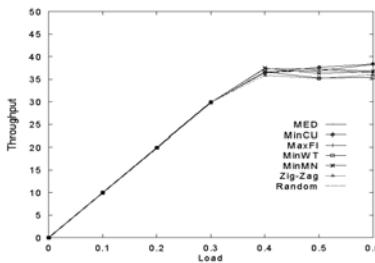


Fig. 4. Net. throughput for hotspot traff.

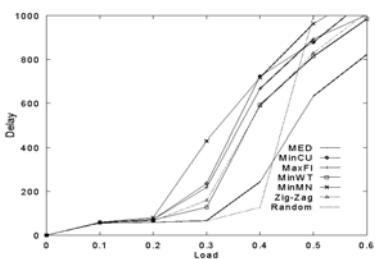


Fig. 5. Net. delay for trans. traff.

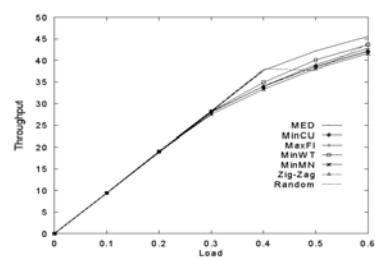


Fig. 6. Net. throughput for trans. traff.

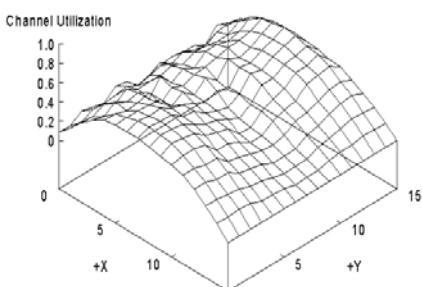


Fig. 7. Chan. util. map for MED under unif. traff.

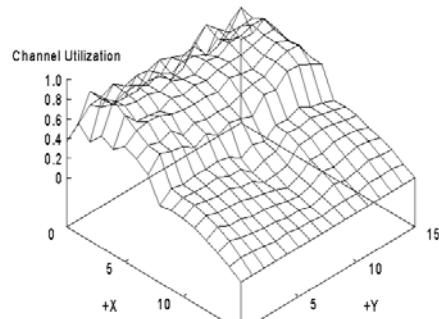


Fig. 8. Chan. util. map for MED under hotspot. traff.

Minimizing Lightpath Set-Up Times in Wavelength Routed All-Optical Networks

M. Shiva Kumar and P. Sreenivasa Kumar

Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai 600 036, India
shiva@aries.iitm.ernet.in,psk@iitm.ernet.in

Abstract. Wavelength Division Multiplexing (WDM) technology evolved to effectively utilize the enormous bandwidth of optical fiber. In circuit-switched WDM networks *lightpaths* are used to transfer data between a given pair of nodes. A lightpath is an all-optical channel established between the given pair of nodes by configuring the Wavelength Routing Switches (WRS) at the intermediate nodes along the path. In a WRS, the time required to reconfigure the switch is often long when compared to the speed of data passing through the switch, and it has significant impact on the performance of the network. In this paper, we present a new technique to mitigate the lightpath set-up time overheads in Dynamic Lightpath Establishment (DLE) algorithms for wavelength continuous circuit switched all-optical networks. Simulation studies indicate that the new technique significantly reduces the number of reconfigurations required to set-up new lightpaths.

1 Introduction

Wavelength division multiplexing (WDM) is a promising technology to effectively utilize the enormous bandwidth of optical fiber by overcoming the electronic bottleneck of slow switches[2]. In this technology the transmission spectrum of a fiber link can be divided into many protocol transparent channels where each channel can be operated at peak electronic speed. A WRS is an optical switch that is capable of routing a signal based on its wavelength and the input port. Using WRS, each wavelength on any input fiber can be interconnected to any output fiber provided that output fiber is not already using the same wavelength. In wavelength routed networks, in order to transfer data from one access station to another, a connection needs to be set-up at the optical layer similar to the case in circuit-switched telephone network. This operation is performed by determining a path in the network connecting the source node to the destination node and by allocating a single free wavelength on all of the fiber links in the path. Such an all-optical path is referred as *lightpath*[1]. The intermediate nodes in the path route the lightpath in the optical domain using WRS. A fundamental requirement in a wavelength-routed optical network is that two or more lightpaths traversing the same fiber link must be on different wavelengths so that they do not interfere with one another.

Given a set of lightpaths, we need to route and assign wavelength to each of them; this is called the Routing and Wavelength Assignment (RAW) problem. In this paper we consider a dynamic RAW problem in which lightpath requests arrive randomly with random call holding times[3][1][4]. In this, call blocking probability and congestion of the network will depend on the order in which connections arrive to the network.

In DLE, routes for requested lightpaths can be chosen either from a pre-computed set of paths between the $s-d$ pair (*Fixed-Alternate Routing (FAR) or Static Routing*)[5] or by using network state information to find the shortest path (*Adaptive Routing or Dynamic Routing*)[3]. In FAR each $s-d$ pair is assigned a set of paths. This set may be searched in a fixed or adaptive order to find an available path[5]. A call is blocked if none of its associated paths is available. In adaptive routing algorithm, route and wavelength for the requested lightpath are determined dynamically by considering the current network state. In order to find routes dynamically, a *layered-graph* is created. The layered-graph is a directed graph obtained from the given physical network topology $G = (N, L)$, by replicating it as many times as there are wavelengths in the network. The algorithm finds a shortest path on each layer, and then chooses the least costly one among them.

The present RAW algorithms do not consider the existing configuration of a switch before establishing a lightpath. In a reconfigurable optical switch, the time required to reconfigure the switch is often long when compared to the speed of data passing through the switch. It is important to consider the delay introduced by slow switches in DLE where the lightpath requests come in real time. In this paper we develop a new technique to minimize the lightpath set-up times by reusing the switch configurations at intermediate nodes of the lightpath.

In Section 2 we propose the new technique to minimize lightpath set-up times in wavelength routed all-optical networks. Section 3 presents the algorithms and simulation details. Section 4 presents numerical results and the paper concludes in Section 5.

2 Lightpath Routing Using CST

In this section, we develop a new technique to minimize lightpath set-up times in all-optical networks by using switch status stored at Connection Status Table (CST) in dynamic lightpath establishment. We observe that at the time of setting up a new lightpath, it is possible to reuse the current configuration set-up of the WRS rather than requiring that all the intermediate WRS be reconfigured. Once a certain input-output permutation has been set-up at a WRS, we assume that the connections persists till the switch is reconfigured. Thus if a new connection can make use of the current configuration of a WRS, we can avoid the overhead of reconfiguration. In order to keep track of the status of the WRS, we introduce three states called *up*, *reserved*, and *null* for a connection and maintain the information in the CST of each WRS. The switch information is updated when a connection is established or taken down. At the switch, for a

particular wavelength k , the connection from an input port A to an output port B may be in one of the following states: *up*, *reserved* or *null*.

- *up* : indicates connection is presently used by a lightpath.
- *reserved* : indicates connection does not correspond to any active lightpath in the network, instead to some old connection.
- *null* : none of the input fibers are connected to any of the output fibers. We assume all the WRS settings at initial stage are in *null* state.

The controller keeps a connection status table that indicates the settings of its switches (which of the input ports are connected to which of the output ports of a WRS and on which wavelengths), the connection-id of the connections that use the WRS and their state. Additional details can be found in[6].

Simulation Details : To compute the number of switch settings required to establish a lightpath, we represent the status of a WRS by using $|A|$ vectors at each node. The size of each vector is equal to the number of input/output ports of the switch. The vector of wavelength λ_i gives the information of which input ports are connected to which output ports on λ_i . In order to establish a lightpath on the given route and wavelength, the algorithms find the switch configuration of intermediate WRSs with the help of the vectors. If the current configuration is appropriate for the requested lightpath, the algorithm checks the status of the connection at CST. If the status of a required connection at an intermediate WRS is *null*, the algorithm reconfigures that WRS without affecting the other connections and changes the state of the connection from state *null* to state *up* at the CST of that node (WRS). If the state of the required connection is *reserved*, it indicates that required connection is currently not used by any other lightpath in the network. In this case, the algorithm simply changes the status of the connection from *reserved* to *up* at CST of that node.

The modified fixed-alternate routing and adaptive routing algorithms are given in Table 2 and Table 3, respectively.

3 Numerical Results

In this section we describe the simulation results obtained for 28 node USA long haul network. Table 1(a) shows the numerical results for modified FAR and existing algorithm. We ran our simulation for 20,000 calls in case of FAR and 10,000 calls in case of adaptive routing. The load is denoted in Erlangs, where an Erlang is defined to be the number of calls per unit call holding time. The third and fourth columns show the number of calls serviced by proposed algorithm and existing algorithm, respectively. The fifth and sixth columns show the number of switch settings required in order to establish the calls given in third and fourth columns, respectively. There is a significant reduction in the number of switch settings required in the proposed algorithm when compared to the present routing algorithms. Table 1(b) shows the numerical results for adaptive routing algorithm. Here also, we observe substantial reduction in the number of switch settings required by the proposed algorithm.

Table 1. Simulation results for 28 node USA long haul network with 8 wavelengths
 (a) fixed-alternate routing with three alternate routes (b) adaptive routing

Fixed Alternate Routing								Adaptive Routing							
<i>load</i>	#calls serviced by		#settings required for		% Gain			#calls serviced by	#settings required for	% Gain					
	new	current	new	current											
	Algorithm	Algorithm		Algorithm				Algorithm	Algorithm						
10	20000	20000	23071	88090	73.80			10000	10000	4164	24637	83.09			
20	20000	19999	23554	88123	73.27			10000	10000	4088	24645	83.41			
30	20000	19994	27031	88548	69.47			10000	10000	4482	24689	81.84			
40	19981	19892	31351	89260	64.87			10000	10000	5124	24925	79.44			
50	19871	19499	33867	88300	61.64			10000	10000	6017	25371	76.28			
60	19596	18894	34870	85716	59.31			10000	10000	69984	19987	6922	26233	73.61	
70	19084	18135	34752	81491	57.35			10000	10000	7925	19932	7944	27370	70.97	
80	18261	17354	34051	76838	55.68			10000	10000	8902	278029	68.59			
90	17541	17037	32595	72665	55.14			10000	10000	9402	9415	8925	27713	67.79	
100	16701	16420	31440	68708	54.21			10000	10000	8958	8962	8659	26731	67.06	

4 Conclusions

In this paper, we proposed a new technique to minimize lightpath set-up times in wavelength routed all-optical networks by using switch status stored in the Connection Status Table at the nodes. We proposed modifications to fixed-alternate and adaptive routing algorithms. The proposed algorithms result in significant reduction in the number of switch settings required compared to the existing algorithms. This in turn leads to substantial reduction in lightpath set-up time. The reduction in the number of switch settings ranges from 65% to 85% for different loads. The blocking probability performance of the proposed algorithm is very close to that of the existing algorithms.

References

1. Imrich Chlamtac, Aura Ganz and Gadi Karmi, *Lightpath Communications: An Approach to High Bandwidth Optical WAN's*, IEEE Transactions on Communications, vol. 40, pp: 1171-1182, July 1992.
2. Paul E. Green et. al., *The Future of Fiber-Optic Computer Networks*, IEEE Computer, vol. 24, pp: 78-87, Sep. 1991.
3. A. Mokhtar and M. Azizoglu, *Adaptive Wavelength Routing in All-Optical Networks*, IEEE/ACM Transactions on Networking, vol. 6, pp: 197-206, April 1998.
4. K. R. Venugopal, M. Shiva Kumar, and P. Sreenivasa Kumar, *A Heuristic for Placement of Limited Range Wavelength Converters in Wavelength Routed Optical Networks*, Proc. of the IEEE INFOCOM '99, 908-915, March 1999.
5. S. Ramamurthy and Biswanath Mukherjee, *Fixed Alternate Routing and Wavelength Conversion in Wavelength Routed Optical Networks*, Department of Computer Science, UC, Davis, USA, Technical Report, March 1998.
6. P. Sreenivasa Kumar and M. Shiva Kumar, *Lightpath Set-Up time Optimization in WDM Networks*, Department of Computer Science & Engg. IIT Madras, India, Technical Report, April 1999.

Table 2. Modified Fixed Alternate Routing Algorithm

```

call_establish( $s, d, id$ )
/* establish a lightpath from  $s$  to  $d$  */
begin
   $j \leftarrow 1$  /* counter for selecting the route
    from the routing table */
  do
    begin
      for each  $\lambda_i \in \Lambda$ 
         $C(r_{sd}(j)^{\lambda_i}) \leftarrow find\_cost(r_{sd}(j), \lambda_i)$ 
        /* finds the cost of routing the re-
          quest on route  $j$  of  $sd$  pair using
          wavelength  $\lambda_i$  */
      Let  $min$  be the cost of the least costly
        route among the  $C(r_{sd}(j)^\lambda)$ ,  $\lambda \in \Lambda$ 
      if( $(equal(min, \infty))$ ) then  $j \leftarrow j + 1$ 
        /* try the next route */
      else
        begin
          for each  $\lambda_i \in \Lambda$  with  $C(r_{sd}(j)^{\lambda_i})$ 
            equal to  $min$ 
             $S(r_{sd}(j)^{\lambda_i}) \leftarrow compute\_switch_-$ 
               $reconfig(r_{sd}(j), \lambda_i)$ 
        /* computes and returns the number
          of switch reconfigurations required
          to establish a lightpath on the
          route  $r_{sd}(j)$  and wavelength  $\lambda_i$  */
      Let  $\lambda_k$  be the wavelength that requires
        minimum switch reconfigurations
        among the  $S(r_{sd}(j)^\lambda)$ ,  $\lambda \in \Lambda$ 
        for each  $(i, j) \in r_{sd}(j)$ 
           $c(i^{\lambda_k}, j^{\lambda_k}) \leftarrow \infty$ 
      configure_optical_switch( $r_{sd}(j)^{\lambda_k}, \lambda_k$ )
        /* function configures the required
          optical switches along the path
           $r_{sd}(j)$  and updates  $CST$  tables */
      return( $callEstablished$ )
    end
  end
  while( $j \leq M$ )
return( $callBlocked$ )
end

```

Table 3. Modified Adaptive Routing Algorithm

```

call_establish( $s, d, id$ )
/* establish a lightpath from  $s$  to  $d$  */
begin
  for each  $\lambda_i \in \Lambda$ 
     $C(p_{id}^{\lambda_i}) \leftarrow find\_path(s, d, \lambda_i, p_{id}^{\lambda_i})$ 
    /* finds the shortest path  $p_{id}^{\lambda_i}$  on the  $\lambda_i$ 
      subgraph  $G(N^{\lambda_i}, L^{\lambda_i})$  and returns
      the cost of the route on  $C(p_{id}^{\lambda_i})$  */
  Let  $min$  be the cost of the least costly
    route among the  $C(p_{id}^\lambda)$ ,  $\lambda \in \Lambda$ 
  if( $(equal(min, \infty))$ ) then return( $callBlocked$ )
  else
    begin
      for each  $\lambda_i \in \Lambda$  with  $C(p_{id}^{\lambda_i})$  equal
        to  $min$ 
         $S(p_{id}^{\lambda_i}) \leftarrow compute\_switch_-$ 
           $reconfig(p_{id}^{\lambda_i}, \lambda_i)$ 
        /* computes and returns the
          number of switch reconfigurations
          required to establish a lightpath on
          the route  $p_{id}^{\lambda_i}$  and wavelength  $\lambda_i$  */
      Let  $\lambda_k$  be the wavelength that requires
        minimum switch reconfigurations
        among the  $S(r_{sd}(j)^\lambda)$ ,  $\lambda \in \Lambda$ 
      for each  $(i, j) \in p_{id}^{\lambda_k}$ 
         $c(i^{\lambda_k}, j^{\lambda_k}) \leftarrow \infty$ 
      configure_optical_switch( $p_{id}^{\lambda_k}, \lambda_k$ )
        /* function configures the required
          optical switches along the path  $p_{id}^{\lambda_k}$ 
          and updates  $CST$  tables */
      return( $callEstablished$ )
    end
  end

```

Design of Wavelength Division Multiplexed Networks for Delay-Bound Multicasting

C.P. Ravikumar, Meeta Sharma, and Prachi Jain

Department of Electrical Engineering, Indian Institute of Technology
New Delhi 110016, India
rkumar@ee.iitd.ernet.in

Abstract. We address the problem of real-time delay-bounded multicasting in wavelength-division multiplexed networks. to avoid problems of synchronization between video and audio frames We describe a technique to synthesize WDM network topologies that can, with a very high degree of confidence, assure that the multicast traffic is delivered in user-specified limits on time. Unlike existing approaches to WDM network design, we first find a virtual topology that can meet the delay constraints. An embedding of virtual links into physical links is then carried out, followed by an assignment of wavelengths to virtual links. Our design paradigm can accommodate both the design styles that have existed in the past, namely, those that use the wavelength continuity principle and those which make use of wavelength converters. We describe quantitative and qualitative results obtained by using our software tool on several benchmark examples.

1 Introduction

Wavelength Division Multiplexing (WDM) technology offers the capability of building very large wide-area networks consisting of thousands of nodes, with per node throughput of the order of gigabits per second [2,5]. It is the key technology for allowing all end-user equipment to operate at electronic speeds. The fiber can carry a high aggregate system capacity of multiple parallel wavelength channels, with each channel operating at much slower speed.

Multicasting is the simultaneous transmission of data to multiple destinations. Applications such as distance learning, video conferencing, TV-on-demand and remote collaboration require the network to provide efficient and reliable multicast services [3]. In real-time multimedia communication, we also need to address the problems of synchronization and anachronism. The multicast tree for each multicast group should be formed in such a manner that a message originating at a source reaches all the other members of the group within a specified time limit. Such a multicast tree is said to be *delay-bound*.

In Section 2, we present algorithms for virtual topology selection, embedding of the virtual topology into the physical network, and assignment of wavelengths to lightpaths. We shall give the algorithms for both cases, namely, wavelength continuity and wavelength conversion. Algorithm implementation and results on test cases are reported in Section 3.

2 Algorithms

2.1 Problem Formulation

A multicast pattern M_i consists of a source s_i , a set of destination nodes \mathcal{D}_i , and an intensity factor \mathcal{I}_i which is a measure of the bandwidth requirement of the communication. The input to the problem is a set of multicast patterns $\{M_1, M_2, \dots, M_n\}$, along with delay bounds $\{D_1, D_2, \dots, D_n\}$. We assume tree-based routing for each multicast pattern. Routing is carried out in a regular *virtual topology* such as a hypercube, De Bruijn graph, Kautz Graph, etc. The links in the virtual topology are then embedded in the physical network. Such an embedding is referred to as a *lightpath* in WDM literature since a single virtual link may actually get mapped to a *path* in the physical network. We initially assume wavelength continuity i.e. a single wavelength is used to embed a virtual link. Later, we shall relax this restriction and permit the use of wavelength conversion. Wavelength continuity constraint simplifies the problem formulation, but can require the use of a large number of λ . Currently, up to 32 wavelengths can be routed on a single fiber. If we exceed this limit, wavelength conversion becomes necessary. Reducing number of wavelengths can further bring down system cost.

The inputs to the problem are (i) the physical topology, (ii) a description of multicast traffic along with delay bounds for each multicast pattern. The output consists of (i) a virtual topology, (ii) the routing of multicast patterns in the virtual topology, (iii) the embedding of lightpaths in the physical topology, and (iv) assignment of wavelengths to lightpaths assuming wavelength continuity. The objective of the optimization is to reduce the system cost. The constraints in the optimization include (i) delay constraints on multicast patterns and (ii) wavelength continuity requirement. The optimization problem is combinatorial in nature and there are a large number of solutions to the problem. Each aspect of the problem, such as wavelength assignment and routing, are computationally difficult subproblems. For instance, delay-bound multicast routing is in itself a difficult problem, and wavelength assignment is an instance of the graph node coloring problem which is known to be NP-complete. Since the problem in its entirety is unwieldy, one practical approach is to use individual heuristics for each phase in the problem. We use a genetic algorithm to solve steps (i) and (ii), namely, determining a virtual topology and finding multicast trees. We use Dijkstra's shortest path heuristic to embed the virtual paths. We treat wavelength assignment as an instance of the node coloring problem and use the backtracking algorithm to solve the assignment problem.

2.2 Genetic Algorithm

Since routing of multicast patterns cannot be done without knowing the virtual topology (VT), the determination of the best VT that can lead to delay-bound multicast routing is a difficult problem. The problem is further complicated due to additional restrictions that are placed on the structure of the VT. For instance, we can place a constraint on the reliability of the VT. A directed graph structure is said to be k -reliable if there are at least k node-disjoint paths between every pair of nodes. The cost, reliability, and delay success of a VT are closely related. Due to lack of space, we avoid presenting details of the algorithm and refer the reader to [1].

2.3 Wavelength Assignment

The links of the VT found by the genetic algorithm are embedded into the physical topology using shortest path heuristic. Thus, the virtual link (x, y) , where x and y are nodes in the optical network, is mapped to the shortest path between x and y in the physical network. The intuition behind using the shortest path heuristic is to increase the chance of meeting the delay constraints. Each lightpath j so found must be assigned a separate wavelength λ_j subject to the wavelength continuity constraint, which requires that *on each fiber link, no two virtual connections can use the same wavelength*. We construct a graph in which each node represents a lightpath. Two nodes i and j are connected if and only if the embeddings of the lightpaths share a common physical link. We cannot assign the same wavelength to lightpaths i and j if edge (i, j) exists. Thus the problem boils down to one of coloring the nodes of the graph using fewest number of colors. We used a backtracking coloring algorithm for coloring since it generates the optimum solution. Every reduction in the number of wavelengths has a direct bearing on the system cost.

2.4 Wavelength Conversion

The number of wavelengths can be further reduced by permitting wavelength conversion. Converter hardware is also expensive and the tradeoff between the number of wavelengths and the number of converters is guided by the relative cost of a converter and a transceiver. We formulate an optimization problem that starts with the solution found in the steps described above and “eliminates” wavelengths through the placement of the smallest number of converters. Let $[A_{ij}]$ denote a 0/1 matrix of size $e \times w$, where e is the number of physical links and w is the number of wavelengths found above. A_{ij} is set to 1 or 0 depending on whether or not the link i has wavelength λ_j passing through it. We show a heuristic for the problem in Figure 1.

3 Implementation and Results

We used our software to design a WAN the 14-node NSFNET (National Science Foundation Network) which is an example used in the literature [4]. VTGEN was tested for various multicast patterns (see Table 1). Virtual topologies obtained for these multicast patterns have been illustrated in figure and 2. Table 2 shows that in each case it was possible to find a virtual topology which satisfied the delay bounds for all multicast patterns. As expected, the number of links in the virtual topology and the number of wavelength assigned increased with the number of multicast groups. The results of wavelength conversion are given in Table 3. We compare our results with those in [4] (Table 4).

4 Conclusions

We proposed a new methodology for topological design of WDM networks considering the issue of real-time multicasting on lightwave networks. We have used our algorithms

```

procedure WavelengthConversion( $A, e, w$ )
begin
    Initialize array  $\text{Mark}[1:w]$  to all 0.
    Set  $\text{Mark}[j]=1$  if  $A_{ij} = 1 \forall i$ 
    // If  $\text{Mark}[j] = 1$ , then  $\lambda_j$  is “essential”
    Remove all columns with  $\text{Mark}[j]=1$ 
    if number of columns left is 1 then Stop
    while (not all columns are marked) do
        Let  $k$  be the column with least number of 1’s
        Construct a set  $S_k$  of all columns which are compatible to column  $k$ 
        //  $r$  and  $t$  compatible if  $\exists$  no  $i$  s.t.  $A[i][r] = 1$  and  $A[i][t] = 1$ 
        if ( $S_k \neq \text{NULL}$ )
            for all  $j \in S_k$ 
                Merge  $k$  with a column  $j \in S_k$  s.t. # of 1’s in  $j$  is maximum
                print "use  $\lambda_j$  instead of  $\lambda_k$ "
            end forall
        end if
    end while
end

```

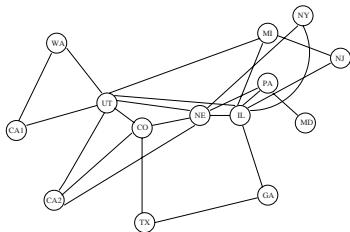
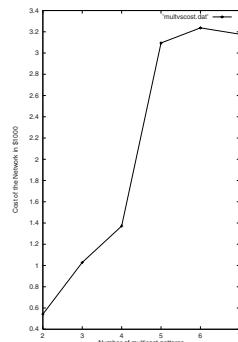
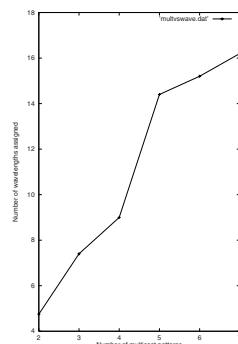
Fig. 1. Algorithm for Wavelength Minimization

to design a wide-area lightwave network of 14 nodes that correspond to the network sites for the NSFNET (National Science Foundation network, USA). Excellent results were obtained for all the examples we tested our software on. The number of virtual links in the virtual topology varied from 20 to 50 as we varied the number of simultaneous multicasts from 2 to 8. In all the cases, the delay-success confidence index varied in the range 0.75 to 1.0. A net reduction as high as 20% in network cost was obtained by the placement of wavelength converters.

References

1. Ajit Dash. *Delay Bound Multicasting* M.Tech Thesis, Dept. of Math., IIT, Delhi, 1996.
2. P.E. Green, Jr. *Fiber Optic Networks*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.
3. V.P.Kompella, J.C.Pasquale. *Multicast Routing for Multimedia Communication*, IEEE/ACM Trans. on Networking, Vol. I, No. 3, June 1993.
4. B.Mukherjee, S.Ramamurthy, D.Banerjee and A.Mukherjee. *Some principles for designing a wide area optical network*, IEEE INFOCOM '94, 1994 .
5. R.Ramaswami and K.N.Sivarajan, *Routing and Wavelength Assignment in all-optical Networks*, IEEE/ACM Trans. on Networking, Vol. 3, No. 5, October 1995, Pages 489-550.
6. Z.Zhang and Acampora. *A heuristic wavelength assignment algorithm for multihop WDM networks with wavelength routing and wavelength re-use*, IEEE/ACM Trans. on Networks, Vol. 3, No. 3, Pages 281-288, June 1995.

S.No.	Source	Destinations
1	CA1	WA, TX, CO, NJ
2	UT	CA2, NE, IL, MI
3	NY	NE, GA , IL, PA
4	TX	UT, GA, NJ, PA
5	NE	NY, CA2, IL, CO
6	MD	WA, UT, PA
7	IL	MI, NJ, UT, GA

Table 1. Example Multicast Patterns**Fig. 2.** Virtual Topology for patterns of Table 1**Fig. 3.** Variation of the cost with # of multicasts

# Mcast	Links	Cost	# λ	DSP
2	17	0.54	5	1.0
3	24	1.02	7	1.0
4	26	1.37	9	1.0
5	45	3.09	14	1.0
6	47	3.23	15	1.0
7	48	3.17	16	1.0
8	48	3.4	17	1.0

Table 2. Results for different patterns

# λ	Cost (M\$)	Conv.	Cost after conversion
16	2.95	3	2.58
16	3.09	2	2.69
18	3.37	7	2.82
20	4.69	4	4.41

Table 3. Effect of λ conversion on cost

# λ	Cost	Cost in [4]
4	.43	1.45
5	.49	1.57
6	.67	1.69
7	.88	1.45
8	.99	1.51

Table 4. Comparison of Network Equipment Cost)

Generalized Approach towards the Fault Diagnosis in Any Arbitrarily Connected Network

Bhaskar DasGupta, Senior Member, IEEE; **Sudip DasGupta**; **Professor Atal Chowdhury**, Department of Computer Science, Jadavpur University, India.

Abstract - The present work is a realistic attempt to achieve an efficient and truly distributed generalized fault diagnosis algorithm for identifying faulty units in any arbitrarily interconnected network. Each node starts the testing procedure simultaneously. The algorithm is based on message passing between the members of the network using existing communication links and depending upon the response from receiving members. A sender member infers about the faulty components of the network in the form of a bit pattern (the bit pattern provides one bit for each processing unit), which is formed, processed and stored independently in each unit. The approach is fully parallel and distributed with respect to all members in the system and exploits fully the inherent parallelism of any interconnected system by activating all its nodes and links simultaneously. The algorithm guarantees proper faulty unit detection so long as the system remains connected even in the presence of faults i.e. the number of faulty processors should not exceed the node connectivity of the network. The main attractive feature of the algorithm is that fault-free node can diagnose the faulty units independently without the help of any central unit. The time required to perform the diagnosis is proportional to the number of processing units in the system, i.e. the time complexity of the algorithm is N , where N is the number of nodes in the connected network.

Index terms: System topology, distributed fault detection, system graph diameter, message passing.

I. Introduction

The vast development of VLSI technology has invited the concept of fast, reliable and powerful computing systems by interconnecting a number of conventional processors as replacement for the uni-processor system. Since the processing, control, and database are distributed in these systems, they do enjoy certain natural advantages over the centralized system from the reliability viewpoint. Such interconnected distributed multiprocessor architecture is the natural requirement for many critical application areas as well as in conventional application areas. Reliability can be achieved by such architectures through redundancy of processing resources. The multiplicity of processing elements is exploited to speed up by parallelism and to gracefully tolerate the failure of one or more processing elements without affecting system performance to a large degree. Meanwhile, it becomes an urgent task to develop practical techniques for fault diagnosis of such systems.

An important aspect of a distributed system is the interconnection topology. There are well-defined relationships between the system topology and the message delay, the routing algorithm, fault tolerance and fault diagnosis. Various researchers have proposed several different topologies, some of which have been implemented in appropriate application areas. All of these structures have attractive features as well as inadequacies.

Fault diagnosis schemes, which are available today, are tightly coupled with the topology of the system – loop , hypercube, star, etc.. In 1982 Reddy and Pradhan suggested a distributed fault detection¹ scheme, which was specific to the system topology proposed by them. Moreover, in their proposed fault detection algorithm, a

concept of root node emerged. It initiated a top-down approach for testing and the test results were transmitted bottom-up. Root node examined the test result and relayed it to all the nodes in the system. Hence, such a distribution extended only to the root nodes of the possible testing graphs.

In this work, a new approach towards fault detection is presented. This is a generalized algorithm for parallel detection of faults and independent of the network topology. So it can be easily applied to any standard architecture (e.g. loop, star, hypercube, etc.), as well as to any arbitrary interconnection of the processors.

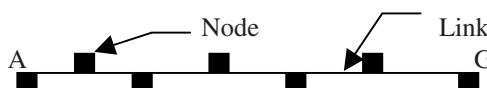
The algorithm is based on the following assumptions:

1. The number of processor faults, at any instant of time, will be within tolerable limits of the network, i.e. at no time the number of faults will exceed the node connectivity of the system graph. Hence, the network never gets disconnected.
2. New faults will not be generated during the execution of the fault diagnosis algorithm. This is not a rigid restriction.
3. Only permanent faults are considered - transient fault detection is not guaranteed.

II. Distributed Fault Diagnosis

For the purpose of fault detection, a system is represented by a Graph $G=(V, E)$ whose vertex or node (V) denotes processing units in the system and edges (E) denote communication links between units in the system. The status of each unit may be faulty (denoted by 0) or fault free (denoted by 1). At the very beginning, each non-faulty node in the system graph determines the status of its immediate neighbors (whether a particular neighbor is faulty or not). It uses a N bit status vector (which is initially set to all zeros) for this purpose, where each bit corresponds to one of the nodes in the system graph and N represents the total number of processors in the system. Each non-faulty node toggles its own bit in the status vector, as well as the bits corresponding to all its non-faulty neighbors.

To know the fault status of the neighbor, it sends a request to all neighbors and initiates a time out (the time out interval is determined by the maximum delay of communication between any pair of neighboring nodes in the system graph). If a neighbor responds within the time out interval, then it is assumed to be healthy else it is treated as faulty. So after the time out, each node in the system graph has the health information about its immediate neighbors (i.e. those nodes, which are at present at a minimum distance of one apart from the node under consideration). This is how the initial status vector is formed.



A linear structure/ Number of hops in between node A and G is six i.e.(N-1)

Fig. 1

The longest distance, in terms of hops, between each pair of nodes in a linear structure, as in **Fig.1**, is $(N-1)$. Each node gathers the health information of its immediate neighbor in the beginning. So each node goes parallel for $(N-2)$ iterations,

where N is the number of nodes in the system. After p^{th} iteration, every node has information about the nodes at a minimum topological distance of exactly $(p+1)$. This means that with the first iteration, a node has information regarding the nodes' neighbor to its neighbors. In the next iteration it comes to know about the status of the nodes which are at a distance two from its neighbors, and so on.

With each iteration, the following two actions are executed in parallel:

- 1) A node sends a request to all its' non-faulty neighbors for their status vector and in turn receives a vector from each. Each of these vectors has one bit for each node in the system. In such a vector, if node k has been inferred as non-faulty by the node sending this vector, then the k^{th} bit (corresponding to node k) is set to 1. Otherwise the corresponding bit is set to 0. These vectors, received from its non-faulty neighbors, are **OR**'ed together along with its' own status vector, obtained at the beginning. This **OR**'ed value is used as the new status vector for the next iteration.
- 2) The node transmits, upon requests from all the non-faulty neighbors, the **OR**'ed vectors obtained from the last iteration.

After a finite number of iterations, all the nodes have a consistent fault pattern in their respective status vector. The final status vector has 1's at the positions corresponding to the non-faulty nodes in the system and 0's corresponding to the faulty ones.

III. Formal Statement of the Algorithm

For each node we have associated three N bit vectors namely **F**, **L** and **C**. The definition of each vector is given below.

F: The status vector formed at the beginning, which stores the conditions of each of its neighbors.

C: Stores the value of the current status vector **OR**'ed with the status vectors received from the neighbors from a particular iteration.

L: Stores the value of **C** at the end of each iteration, and passing this vector to the neighbors during the immediate next iteration. (At the beginning **L** assumes the value of **F**.)

The following convention is used in defining the steps of the algorithm.

Fi, Li, Ci : The vector F, L & C respectively corresponding to node i.

Fij, Lij : The j^{th} bit in status vector Fi and in Li vector respectively.

The following steps denote the fault diagnosis algorithm for the node i.

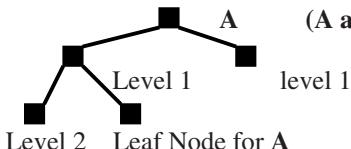
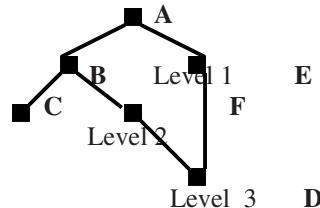
1. **Fi := 0**
2. **Fii := 1**
3. **For each node j, which is neighbor of node i**
 - Do 3.1 thru 3.3 in sequence**
 - 3.1 **Send request to node j**
 - 3.2 **Initiate time out**
 - 3.3 **If (node j responds within time-out)**
 - Then set $F_{ij} := 1$**
 4. **Li := Fi; Ci := Fi;**
 5. **Repeat (N-2)times steps 5.1 thru 5.2 in sequence**
 - 5.1 For each node j, which is non-faulty neighbor of node i, do steps 5.1.1 and 5.1.2 in parallel.**

5.1.1 Do in sequence**5.1.1.1 Request neighbor node j for Lj****5.1.1.2 Ci := (Ci OR Lj)****5.1.2 Send Li to neighbor node j****5.2 Li := Ci****6. Stop**

After (N-2) iterations, all non-faulty processors in the system will get the consistent and final fault pattern of the system. However, this is only true for worst case (e.g. for a faulty node or link in a loop structure). If all the nodes in the system are fault free, the maximum number of iterations needed for a node is (Diameter-1) as Diameter of a system graph is the longest distance between any pair of nodes in the system to infer. However, some of the processors might be able to obtain the final pattern earlier, i.e. before completion of (D-1) iterations. Hence, we claim that:

Theorem :

"If the status vector Li of node i remains unchanged for two consecutive iterations, it will never change again."

**Fig. 2**

Node D can be reached from node A either via path A-B-F-D or path A-E-D.

Fig. 3**Proof:**

At the beginning, a node forms its status vector making the corresponding bits of its fault free neighbors high. After that, in each iteration information of the nodes are pumped up to the node and by **OR**'ing the corresponding bits in its \mathbf{L} vector is made high/low according to the health of the tested nodes. Hence, in any iteration it receives information of the nodes, which are at a level next to the nodes it tested in the previous iteration in the testing graph. Its \mathbf{L} vector changes in each iteration until it receives the information of the leaf nodes or the nodes that are at the longest distance in the testing path. As the leaf nodes are the terminal nodes in the system graph, it is certain that its \mathbf{L} vector remains unchanged in the following iterations as no new nodes are encountered further. If there is more than one path to reach the node furthest from the testing node, then after receiving the information of the highest level node it will receive the information of a node which is at one level lower than the node it just tested in the previous iteration. As in **Fig. 3**, after receiving the information of node D it will receive the information of node F in the next iteration. But the information of node F is already with the testing node as it is at a level lower than the highest level i.e. D (through other path (A-E)) and it already made the corresponding entry in its \mathbf{L} vector for that node. Hence, here also the \mathbf{L} vector

remains unchanged in the following iterations. Therefore we can say, if any node detects no change in its **L** vectors in two consecutive iterations, it should be considered to be the final **L** vector containing information of all the nodes in the system. \square

The node may initiate a STOP command and then broadcast the final status vector to its neighbors and the neighbors in turn can pass it on to their neighbors and so on. On receiving the STOP instruction from its neighbor a node stops execution of its fault detection routine and updates its **L** vector to the bit pattern it receives from its neighbor as the final **L** vector. In view of this observation the modified algorithm is given below:

In order to implement this we require an extra N bit vector **PrevL** for each node.

PrevLi: Stores the value **Li** of previous iteration and at the very beginning it assumes all the bits to be 1.

Thus the final algorithm is:

1. **Fi := 0**
2. **Fii := 1**
3. **For each node j, which is neighbor of node i**
 - Do 3.1 thru 3.3 in sequence**
 - 3.1. **Send request to node j**
 - 3.2. **Initiate time out**
 - 3.3. **If (node j responds within time -out)**
 - Then set Fij := 1**
4. **Li := Fi; Ci := Fi ; PrevLi := all 1's;**
STOP := 1;
5. **While(STOP) Do 5.1 thru 5.4 in sequence**
 - 5.1. For all non-faulty neighbors (node j) of node i, do step 5.1.1 and 5.1.2 in parallel.**
 - 5.1.1. Do in sequence**
 - 5.1.1.1. Request neighbor node j for Lj**
 - 5.1.1.2. Ci := (Ci OR Lj)**
 - 5.1.2. Send Li to neighbor node j**
 - 5.2. Li := Ci**
 - 5.3. STOP := (Li XOR PrevLi)**
 - 5.4. PrevLi := Li**
6. **Stop**

IV. Illustration

Let us consider an arbitrarily connected network to demonstrate the effectiveness of our proposed algorithm for proper fault detection. The network interconnection with eight nodes is shown in **Fig.4**. We consider **node 7** as a faulty one.

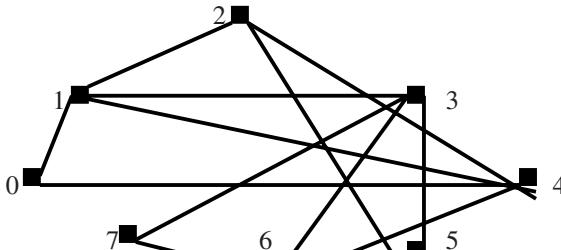


Fig. 4

DETECTION OF NODE FAILURE

Generated status vector F of each node

As node 7 is faulty, each node sets the bit corresponding to node 7 as 0 in its status vector.

	7	6	5	4	3	2	1	0
Node[0].F =F0=	0	0	0	1	0	0	1	1
Node[1].F =F1=	0	0	0	1	1	1	1	1
Node[2].F =F2=	0	0	1	1	0	1	1	0
Node[3].F =F3=	0	1	1	0	1	0	1	0
Node[4].F =F4=	0	1	0	1	0	1	1	1
Node[5].F =F5=	0	1	1	0	1	1	0	0
Node[6].F =F6=	0	1	1	1	1	0	0	0
Node[7].F =F7=	0	0	0	0	0	0	0	0

Generated L vectors of each node

Iteration No: 1

	7	6	5	4	3	2	1	0
Node[0].L=L0=	0	1	0	1	1	1	1	1
Node[1].L=L1=	0	1	1	1	1	1	1	1
Node[2].L=L2=	0	1	1	1	1	1	1	1
Node[3].L=L3=	0	1	1	1	1	1	1	1
Node[4].L=L4=	0	1	1	1	1	1	1	1
Node[5].L=L5=	0	1	1	1	1	1	0	
Node[6].L=L6=	0	1	1	1	1	1	1	1
Node[7].L=L7=	0	0	0	0	0	0	0	0

Iteration No: 2

	7	6	5	4	3	2	1	0
Node[0].L=L0=	0	1	1	1	1	1	1	1
Node[1].L=L1=	0	1	1	1	1	1	1	1
Node[2].L=L2=	0	1	1	1	1	1	1	1
Node[3].L=L3=	0	1	1	1	1	1	1	1
Node[4].L=L4=	0	1	1	1	1	1	1	1
Node[5].L=L5=	0	1	1	1	1	1	1	1
Node[6].L=L6=	0	1	1	1	1	1	1	1
Node[7].L=L7=	0	0	0	0	0	0	0	0

As node 1, 2, 3, 4 and 6 find that the **L** vectors formed in two consecutive iterations, i.e. in iteration 1 and 2 are same, they can issue the STOP command to conclude that node 7 is faulty as the bit corresponding to node 7 is 0. Hence, the algorithm can detect specific faulty node.

V. Conclusion

Here we have shown a simple distributed fault detection algorithm, which can be applied to any arbitrarily connected network. The algorithm ensures that all the nodes execute the same sequence in parallel, by invoking all links of each node and after a finite number of iterations all non-faulty nodes gets identical fault patterns. As such, there is no central co-ordinating node, but the algorithm requires one node to be assigned for initiating the fault diagnosis algorithm at regular intervals. Thus the requirement for a co-ordinator election algorithm is not completely ruled out. Along with message passing, if the concept of the majority voting is introduced, the present work also can be extended to detect transient faults.

References

- [1] Pradhan D. K. and Reddy S. M., „A fault tolerant communication architecture for distributed systems“, IEEE transactions on Computers, Vol. C-31, September 1982, pp. 863 – 869.
- [2] Kuhl J. and S. M., „Distributed fault tolerance for large multiprocessor systems“, In Proc. 7th. Annual symposium on Computer Architecture, May 1980, pp. 23–30.
- [3] Wilkov R. S., „Analysis and design of reliable computer networks“, IEEE transactions on Communication Technology, Vol. COM-18, October 1970, pp. 501 – 519.
- [4] Armstrong J. R. and Gray G. G., „Fault diagnosis in a boolean n-cube array of multiprocessors“, IEEE Transactions on Computer, Vol. C-30, August 1981, pp. 587 – 590.
- [5] Ahmed EL-Amawy and Shahram Latifi, „Properties and performance of folder hypercube“, IEEE Transactions on Parallel Distributed System, Vol. 2, No. 1, January 1991, pp. 31–41.
- [6] Xiaofan Yang, Tinghuai Chen, Zehan Cao, Zhongshi He and Hongqing Cao, „A new scheme for the fault diagnosis of multiprocessor systems“, Proceedings of the Fifth Asian Test Symposium, 1996., pp. 289-294.

Author Index

- A.A. Abouzeid 309
S. Aluru 339
R. Andonov 125,274
S. Arvindam 61
V. Ashok 349
G. Athithan 45
M. Azizoglu 309
- T.C. Babu 349
S. Balaji 381
S. Balev 274
P. Banerjee 202
D. Bansal 246
G. Baumgartner 103
F. Belkouch 181
V.K. Bhargava 219
P.S. Bhattacharjee 322
R.V. Boppana 239
A. Boukerche 189
H. Bourzoufi 125
M. Bui 181
- V. Chaudhary 77,87
P.P. Chaudhuri 269
A. Choudhary 354
D.R. Choudhury 269
A. Chowdhury 404
L. Chen 181
K.W. Chin 229
D. Cociorva 103
- D. Dalton 364
S. Damodaran 234
D. Das 143
P. Das 143
S.K. Das 189,291
B. Dasgupta 404
P. Dasgupta 143
S. Dasgupta 404
- A.K. Datta 181
T. Decker 261
P.S. Dhekne 66
- A. Fabbri 189
C. Farrell 229
P. Foglia 133
B. Folliot 71
M. Franklin 28,33
A. Freville 274
N. Futamura 339
- R. Giorgi 133
K. Gopinath 13
D. Goswami 38
R. Govindarajan 111
P. Gupta 212
- Y. Hajmalmoud 71
A.Z.M.E. Hossain 219
- J. In 331
J. Irwin 21
- P. Jain 399
Y. Jiang 207
C. Jin 331
J. Ju 87
- N.K. Kakani 291
M. Kalia 246
M. Kang 373
H.K. Kaura 66
A. Khaleel 55
S.K. Kodase 158
S.P. Konduru 239
W. Krandick 261
G. Kumar 169
M. Kumar 229
P.S. Kumar 394

- C.-C. Lam 103
A. Laszloffy 359
A. Lele 299
C.H.C. Leung 207
T.J. Li 389
W.-k. Liao 354
J. Lin 120
K.H. Liu 207
X. Liu 389
J. Long 359
- S. Mahajan 66
H. Mahanta 212
S. Majumdar 151
R. Mall 38,95,158
A. Mandal 45
G. Manimaran 381
M.K. Marina 239
M.A. Marsan 315
D. May 21
M. Meo 315
W. Mohan 28
A. Mukherjee 322
H.L. Muller 21
C.S.R. Murthy 381
- K.S. Nandakumar 13
S.K. Nandy 299
N. Narang 169
- S. Olariu 284
- D. Page 21
A. Pal 158
L.M. Patnaik 279
A.K. Patra 359
K. Paul 269
J. Peir 331
M.C. Pinotti 284
S. Pokarna 61
C.A. Prete 133
H. Praveen 61
- R. Qiao 120
K. Rajan 279
- K. Rajesh 66
V.J. Ramanan 111
D. Ranjan 339
S. Ranka 331
B.S. Rao 45
C.P Ravikumar 169,399
A.L.N. Reddy 55
D.A. Reimann 77
S. Roy 309
- P. Sadayappan 103
A. Saha 66
D. Saha 322
S. Sahni 331
H Saran 246
N.V. Satyanarayana 158
S.K. Sen 291
P. Sens 71
I.K. Sethi 77
M. Sharma 45,399
W. Shi 163
M. Shiva Kumar 394
J.F. Sibeyn 197
B. Sidi-Boulenouar 125
K.M. Sivalingam 234
- Z. Tang 163
C. Timsit 3
J. Trdlička 253
P. Tvrdík 253
- S. Vadlapatla 33
P. Varshney 354
- D. Weiner 354
- Y. Xiang 389
- O. Yildiz 189
C. Yu 373
Y. Yuan 202
- S. Zertal 3
Z. Zhang 120
S.Q. Zheng 284
N. Zhu 120